no. 640

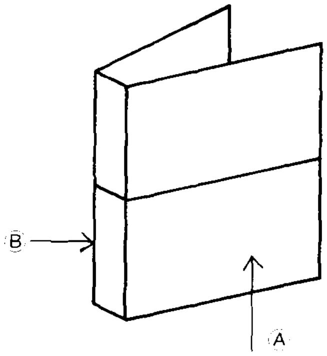
mecc

minnesota educational computing consortium

files on the apple

training materials
for the **apple**® **II** computer

How to use your MECC folder cover inserts



Tear cover on perforated lines. Place half-page title section in the clear vinyl pocket Ⓐ on the front of the binder. Insert finger-sized title in the clear plastic pocket on the spine Ⓑ.

files on the apple

files on the apple

mecc
minnesota educational computing consortium

files on the apple

training materials
for the **apple**® **II** computer

© Minnesota Educational Computing Consortium
3490 Lexington Avenue North
St. Paul, MN 55112

September 1, 1981
Latest Printing: March 1, 1984

TABLE OF CONTENTS

INTRODUCTION

Beginning programming instruction for students usually focuses on commands and statements that give the computer a logical set of instructions. Eventually it becomes apparent that programs can be made more interesting if the computer applies these instructions to a collection of specified information. At this point, when using BASIC, the READ and DATA statements are introduced. More sophisticated applications using data, however, require greater capability than DATA statements provide, in which case files become the appropriate technique to use. This student textbook is designed to teach users of the Apple II computer the fundamentals of using various types of files.

This book contains five chapters, each of which probably will take a week or two of daily class sessions to cover. Periodic exercise sets are provided in each chapter. A review and review quiz are provided at the end of Chapters 2 through 4. It is assumed that students using this text already have a fairly extensive knowledge of how to operate and program the Apple II. While it is not necessary for them to have background in such special programming features as graphics and sound, it is essential to have good experience with system commands, functions, and logic statements before attempting to learn the use of files.

There are, technically speaking, three types of files used on the Apple:

program files

text files

binary files

The **PROGRAM FILE** is commonly referred to simply as a "program." Because it is assumed that students in a class on files already know how to write a simple program, all BASIC statements will not be covered in this book. Information on beginning programming can be found in the MECC publication Introduction to Applesoft Basic (see Appendix A, "MECC Services").

The **TEXT FILE,** sometimes referred to as a "data file," is used to store information for use by a program. There are two types of text files: **sequential** and **random access.** Both types of text files will be discussed in this booklet.

The **BINARY FILE** is an exact bit-for-bit copy of the information that is stored in a range of the Apple's memory locations. There are two types of binary files: **machine language programs** and **binary data.** Information on machine language files can be found in the MECC publication Apple Assembly Language (see Appendix A, "MECC Services"). Two examples of binary data files are shape tables and numeric data stored in the computer's memory and on a diskette in binary form. Of these only the second kind will be discussed in this book. Information on shape tables can be found in the MECC publications Programmer's Aid Volume 1 and Shape Tables Volume 1, each of which has an accompanying diskette (see Appendix A, "MECC Services").

Answers to all student exercises and review quizzes in this book can be found in a separate Answer Key, available from MECC Distribution.

This book was written by John Arneson, Will Jokela, and Tom Prosen of the MECC Instructional Services staff.

CHAPTER 1

DEFINITIONS AND REVIEW OF SYSTEM COMMANDS

1.1 FILE TYPES

Text Files

A text file is a storage area located on a diskette. In many ways you can think of a text file as an extended DATA statement. A text file, like a program, is identified by a name. Text files are most often used to store information which will be needed at a later date or to transfer data from one program to another. Text files are broken up into records. A record is considered to be a segment of the file. A carriage return separates the records in the file as they are entered. The items stored in these records are either numeric values or strings made up of letters and symbols.

Sequential Text Files

A sequential file can be used to store both numbers and strings. The word "sequential" is used to denote that the stored information can only be read or written from the beginning to the end of the file. Sequential files should not be used if you plan to read or write randomly in a file. Sequential files can be positioned at a record within the file. This may not be very useful where the record needs to be changed because all the records are of different lengths. For example, if you change a record from 'X' to 'XXX', then you will destroy two characters into the next record. If you need to make changes, then you must write over the file from the point the change was made to the end of the file.

Random Access Text Files

A random access file can be used to store both numbers and strings. Random access files are so named because each record in the file does not have to be read or written in a forward or sequential order. The records can be written in any sequence without disturbing the other records or having to rewrite the entire file. Each record is of the same length, so it is easier to make changes within a file. Random access files should be used when the information in the file will be changed often, when not all the data is needed every time the file is used, and when quick access to a single item of data is important.

Binary Files

A binary data file would be best used where there is a large amount of numeric or string data that is accessed randomly, is accessed repeatedly, and is never changed. A binary data file eliminates the need to access the diskette for each data item, takes up less storage, and is usually faster to use than other data files.

1.2 DEFINITION OF TERMS

There are several computer technical terms used throughout the book. The following list will explain the meaning of each.

RECORD - Files are made up of records. The record length is determined by the programmer when using random access files. The record length in a sequential file is determined by the length of the data put in the record.

FIELD - Records are made up of fields. In a sequential file a record and a field are the same length, so fields are normally referred to instead of records. In a random access file several fields separated by carriage returns can be written in one record.

BYTE - Fields are made up of bytes. The byte is any character of a number or string that is entered into the file. A record byte starts counting with zero at the start of each record. The record byte is used in random access files. The file byte starts with byte zero at the beginning of the file and counts to the end of the file. The file byte is used in sequential files.

BIT - Bytes are made up of bits. The bit is one-eighth of a numeric byte when the numbers are converted into binary data.

TRACK - A track is a segment of a diskette's memory space. A diskette has 35 tracks. Three of these tracks are used for DOS commands and one for the directory, so the user has 31 tracks left for storage.

SECTOR - Tracks are made up of sectors. Each track contains 16 sectors. Out of a total of 560 sectors on a diskette, 496 are available to the user. Each sector contains 256 bytes, for a total of 126,976 bytes of storage available to the user on the diskette.

CONTROL D - Disk Operating System (DOS) commands can be executed within a program. In order to execute such a command you must press the CTRL key and the 'D' key. The ASCII equivalent of a CTRL-D is the function CHR$ with an argument of 4, or CHR$(4). The ASCII equivalent is easier to use in a program. Thus, DOS commands must be preceded by:

```
PRINT CHR$(4)
```

In programs that use GET statements or other ways of inputting, the user may have to precede the Control D with a carriage return, CHR$(13). For example:

```
PRINT CHR$(13);CHR$(4)
```

1.3 REVIEW OF DISK OPERATING SYSTEM (DOS) COMMANDS

DOS commands are used for immediate execution. They are typed in, followed by pressing the RETURN key. The disk drive number referred to can either be 1 or 2. The slot number designating which slot is used for the disk drive controller card can be from 1 to 7, inclusive.

CATALOG

Lists the names of the files on the diskette.

general form: CATALOG,Dd,Ss

where: d is the drive number. Default is the last drive referred to.
s is the slot number. Default is the last slot referred to.

example:

```
CATALOG
CATALOG,D2
CATALOG,D1,S7
```

result:

```
* t s name
```

* if present, indicates the file is locked.

where: t is the type of file.

A	APPLESOFT program file.
I	INTEGER program file.
T	TEXT file.
B	BINARY file.

s is the number sectors that the file uses on the diskette. Each diskette has 496 sectors available for file storage.

name is a legal file name.

example:

```
  T 016 SAMPLE1

* A 008 FIRST SAMPLE
```

LOCK

Locks the file so it cannot be written on.

general form: LOCK name,Dd,Ss
CATALOG (optional)

where: name is a legal file name.
d is the drive number.
s is the slot number.

example: LOCK SAMPLE5
LOCK SAMPLE6,D2
LOCK SAMPLE7,D1,S7

result: *t s name

where: * shows that the file has been locked.

example: *T 020 SAMPLE2

UNLOCK

Unlocks the file so it can be written on.

general form: UNLOCK name,Dd,Ss
CATALOG (optional)

where: name is a legal file name.
d is the drive number.
s is the slot number.

example: UNLOCK SAMPLE8
UNLOCK SAMPLE9,D2
UNLOCK SAMPLE10,D1,S7

result: t s name

example: T 020 SAMPLE2

DELETE

Deletes the file from the diskette.

general form: DELETE name,Dd,Ss

where:
name is a legal file name.
d is the drive number.
s is the slot number.

example:

```
DELETE SAMPLE3
DELETE SAMPLE4,D2
DELETE SAMPLE11,D1,S7
```

result: The file is deleted from the diskette and the name will not be listed in the CATALOG.

RENAME

Renames a file on the diskette.

general form: RENAME name1,name2,Dd,Ss

where:
name1 is the old name.
name2 is the new name.
d is the drive number.
s is the slot number.

example:

```
RENAME SAMPLE3,SAMPLE4
RENAME SAMPLE20,SAMPLE40,D2
RENAME SAMPLE36,SAMPLE50,D1,S7
```

result: The file named SAMPLE3 is now named SAMPLE4.

SAVE

Saves the file on the diskette (strictly used for program files).

general format: SAVE name,Dd,Ss

where:
name is a legal file name
d is the drive number
s is the slot number

example:

```
SAVE SAMPLE3,D2
SAVE SAMPLE4,D2,S7
SAVE SAMPLES
```

result: The program is saved on the diskette and the name appears in the catalog.

Review Quiz - Chapter 1

1. Name the two types of TEXT files.

2. Describe the CATALOG listing:

```
*T 032 SAMPLE
```

3. What command should be used to erase the TEXT file named FILE8?

4. Write the three parts of a file, starting with the largest part and going to the smallest part.

5. How many bits are in one numeric byte on the Apple II?

6. How many sectors are available for files on a diskette?

7. Write the command that would change the name of a file called SAMPLE1 to SAMPLE2.

8. What is the purpose of CHR$(4) in a statement?

9. What DOS command makes it so that a given TEXT file cannot be changed?

CHAPTER 2

SEQUENTIAL FILES

2.1 INITIAL REMARKS

Sequential files are the easiest to use. The order in which you read or write is always in a forward motion, so that once the file has been set up for reading or writing the process will continue to the end of the file. If you plan on positioning to a given field in the file, then you must know the number of the field you want and the number of the field where you are currently located. The carriage return ends each field so that you can keep track of where you are in a sequential file by counting the entries.

The diagram below shows a sequential file with four sets of data. JOHN DOE represents the first and 8 the last. The Ⓡ represents a carriage return, which separates the data. The sets of data are called "fields." The fields are counted starting with zero. Each letter or symbol represents one byte of information.

<table>
<tr><td></td><td>J</td><td>O</td><td>H</td><td>N</td><td></td><td>D</td><td>O</td><td>E</td><td>Ⓡ</td><td>7</td><td>Ⓡ</td><td>M</td><td>A</td><td>R</td><td>Y</td><td></td><td>J</td><td>O</td><td>N</td><td>E</td><td>S</td><td>Ⓡ</td><td>8</td><td>Ⓡ</td></tr>
<tr><td><b>byte:</b></td><td>Ø</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1Ø</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>2Ø</td><td>21</td><td>22</td><td>23</td></tr>
<tr><td><b>field:</b></td><td colspan="8">Ø</td><td></td><td>1</td><td></td><td colspan="10">2</td><td></td><td>3</td><td></td></tr>
</table>

2.2 CREATING A SEQUENTIAL FILE

A file cannot be created by simply entering information directly from the keyboard as when creating a program. A program has to be designed that writes the information on the file. The file commands needed to enter data to a sequential file on a diskette are:

```
OPEN
WRITE
CLOSE
```

Open Statement

general form:

```
ln PRINT D$;"OPEN name"
```

where:

ln is a legal line number.
D$ is a control-D
name is a legal file name.

example:

```
20  PRINT D$;"OPEN FILE1"
30  PRINT D$;"OPEN FILE2"
```

purpose:

The OPEN command is used either to set aside work space for a new file or to access an old file. When it creates a new file, it writes the name of the file in the CATALOG. If the file is a previously created file it positions the file to the beginning of the information. OPEN allocates a 595-byte file buffer to handle the file's input and output.

Alternate Forms of the Open Statement

general form:

```
ln  PRINT D$;"OPEN name,Ss"
ln  PRINT D$;"OPEN name,Dd"
ln  PRINT D$;"OPEN name,Ss,Dd"
```

where:

ln is a legal line number.
D$ is a control-D.
s is the disk drive controller slot number. The number can be from 1 to 7 inclusive.
d is the disk drive number. The number can be 1 or 2.

example:

```
20  PRINT D$"OPEN FILE1,S7"
30  PRINT D$;"OPEN FILE2,D2"
40  PRINT D$;"OPEN FILE3,S7,D2"
```

purpose:

The alternative forms of the OPEN statement are needed when more than one disk drive is used with the Apple. The first form would be used if each disk drive were connected to separate controller cards. The slot used is referred to as s. The second form would be used if two disk drives were connected to the same controller card. The drive used is referred to as d. The last form would be used with more than two drives to keep track of which drive and which controller slot number to use.

Write Statement

general form:

```
ln   PRINT D$;"WRITE name"
```

where:

ln is a legal line number.
D$ is a control-D.
name is a legal file name.

example:

```
23  PRINT D$;"WRITE FILE1"
34  PRINT D$;"WRITE FILE3"
```

purpose:

The WRITE command is used to prepare a file for writing. Once a WRITE command is executed, any subsequent PRINT commands write all characters to the file instead of the screen. The WRITE command can be terminated by any DOS command. The simplest DOS command to use for this is PRINT D$.

Close Statement

general form:

```
ln   PRINT D$;"CLOSE name"
ln   PRINT D$;"CLOSE"
```

where:

ln is a legal line number.
D$ is a control-D.
name is a legal file name.

example:

```
20  PRINT D$;"CLOSE FILE1"
30  PRINT D$;"CLOSE FILE4"
40  PRINT D$;"CLOSE"
```

purpose:

The CLOSE command releases the file from the work space so it can be opened and used again. The CLOSE name statement will close all files with the given name regardless of the drive and slot number. The last example does not use a file name. A CLOSE statement without a file name will close all files that are open.

Try the following sample programs.

problem 1: Create a sequential file that will write the names of the first two months of the year to a file called FILE1.

program 1:
(C2P1)

```
10  D$ = CHR$(4)
20  PRINT D$;"OPEN FILE1"
30  PRINT D$;"WRITE FILE1"
40  PRINT "JANUARY"
50  PRINT "FEBRUARY"
60  PRINT D$;"CLOSE FILE1"
70  END
```

program description:

Line 20 — opens a file called FILE1.
Line 30 — prepares file FILE1 to be written on.
Line 40 and 50 — two fields of alphanumeric data written on the file.
Line 60 — closes the file name FILE1.

After the above program is run, the sequential file would resemble the following diagram:

Diagram 1:	J	A	N	U	A	R	Y	®	F	E	B	R	U	A	R	Y	®
byte:	7								8								
field:	Ø								1								

problem 2: Create a program that will write twelve alphanumeric fields and twelve numeric fields to a file named FILE2.

program 2: (C2P2)

```
10  D$ = CHR$ (4)
20  PRINT "ENTER A NAME AND A NUMBER AFTER EACH"
30  PRINT "QUESTION MARK.  SEPARATE THEM WITH"
40  PRINT "A COMMA."
50  PRINT D$;"OPEN FILE2"
60  FOR X = 1 TO 12
70  INPUT A$,A
80  PRINT D$;"WRITE FILE2"
90  PRINT A$
100  PRINT A
110  PRINT D$
120  NEXT X
130  PRINT D$;"CLOSE FILE2"
140  END
```

program:

Line 50 -- opens a file called FILE2.
Line 80 — prepares file FILE2 to be written on.
Line 90 and 100 -- writes two fields to the file. Each field will be separated by a carriage return.
Line 110 - terminates writing to the file. This is needed so that the question mark from the INPUT statement at 70 will be printed on the screen and not on the file.
Line 130 — closes the file called FILE2.

The run of the above program would write 24 fields on the file, separated by carriage returns.

Diagram 2:	n	a	m	e	1	®	n	u	m	b	e	r	1	®	. .	n	a	m	e	12	®	n	u	m	b	e	r	®

problem 3: Write a program that prints on a file and on the screen.

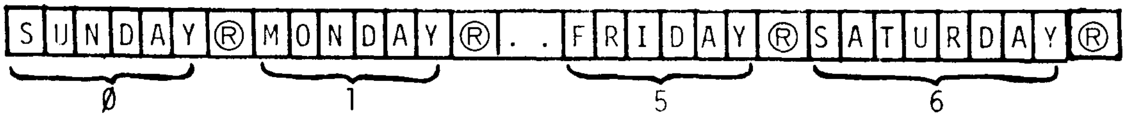
program 3:
(C2P3)

```
10  D$ = CHR$(4)
20  PRINT D$;"OPEN FILE3"
30  FOR X = 1 TO 7
40  PRINT "TYPE IN THE NAME OF DAY ";X;" OF EACH
    WEEK."
50  INPUT A$
60  PRINT D$;"WRITE FILE3"
70  PRINT A$
80  PRINT D$
90  NEXT X
100  PRINT D$;"CLOSE FILE3"
110  END
```

program

Line 20 — opens the file called FILE3.
Line 60 — prepares file FILE3 to be written on.
Line 80 -- the WRITE command is terminated by the control-D so that the next print will not be written on the file.
Line 100 -- closes the file called FILE3.

Diagram 3:


S U N D A Y ® M O N D A Y ® . . F R I D A Y ® S A T U R D A Y ®
Ø
1
5
6

field:

SPECIAL STATEMENTS

MON Statement

general form: PRINT D$;"MON,C,I,O"

where:

C = the file commands will be printed on the screen.
I = the information read from the file will be printed on the screen.
O = the information printed to the file will be printed on the screen.

purpose: To monitor a variety of information: file commands, information taken from the file (INPUT), and information sent to the file (OUTPUT).

NOMON Statement

general form: PRINT D$;"NOMON,C,I,O"

where:

C = the file commands will not be printed on the screen.
I = the information read from the file will not be printed on the screen.
O = the information printed to the file will not be printed on the screen.

purpose: To turn off the MON statement and stop printing of file commands and information taken from and sent to a file. If the MON statement is not turned off it carries over to the next program or command.

Rerun program #3 by adding a MON statement in line 15 and a NOMON statement at line 105:

```
15   PRINT D$;"MON C,O"
105  PRINT D$;"NOMON C,O"
```

The I parameter would have no meaning in line 15 because the program does not require any information to be read from the file; therefore it is left out.

EXERCISE SET 2A

1. What does "sequential" mean?

2. How many bytes would the number 21 take?

3. How many bytes would the name JONES take?

4. Why is the OPEN statement necessary?

5. What is the first field number in a sequential file?

6. Write a program that will write the following names to a sequential file using the first four fields.

   APPLE
   ORANGE
   BANANA
   PEAR

7. Why is the WRITE statement necessary?

8. Write a program that will request from the user the employee list for a company. The number of employees should be requested at the start of the program. Use ten or fewer employees and name the file SAMPLEC2A2.

9. Explain the statement:

```
20  PRINT D$;"MON C"
```

10. There are two statements missing from the following program. What are they and where should they be inserted?

```
10  PRINT D$;"OPEN FILE5"
20  PRINT D$;"WRITE FILE5"
30  READ A$
40  DATA "ONE","TWO","THREE"
50  PRINT A$
60  IF A$ = "THREE" THEN 80
70  GOTO 20
80  END
```

11. Explain the statement:

```
15  PRINT D$;"OPEN FILE16,D2"
```

12. Explain the statement:

```
23  PRINT D$;"OPEN FILE4,S7"
```

2.3 READING A SEQUENTIAL FILE

A sequential file can only be read using a program written to read the information on the file. The file commands needed are:

```
OPEN
READ
CLOSE
```

The OPEN and CLOSE statements were discussed in the last section.

READ Statement

general form:

```
ln   PRINT D$;"READ name"
```

where:

ln is any legal line number.
D$ is a control-D.
name is the file name.

example:

```
20   PRINT D$;"READ FILE6"
30   PRINT D$;"READ FILE7"
```

purpose:

The READ statement is used to prepare the file so that information previously written on the file can be retrieved. Once the READ statement is executed, any subsequent INPUT statements refer to the file. The READ command is terminated by any DOS command. The statement PRINT D$ would be the simplest to use.

Alternative Form of the READ Statement

You can start reading or writing at a given field in the file by using the following file commands:

```
OPEN
POSITION
READ or WRITE
CLOSE
```

The OPEN, READ, WRITE, and CLOSE statements were discussed in a previous section.

POSITION Statement

general form:

```
ln   PRINT D$;"POSITION name, Rp"
```

where:

ln is any legal line number.
D$ is a control-D.
name is the file name.
p is the field position.

example:

```
23   PRINT D$;"POSITION SAMPLE, R10"
40   PRINT D$;"POSITION SAMPLE2,R6"
```

purpose:

The POSITION statement is used to prepare the file at a designated field. After a file has been opened, the file is at the first field, which is the zero (0) field. Therefore, an R10 would be ready to read or write in field number 10, which is the eleventh field in the file. Fields numbered 0 through 9 have been skipped; that is, ten fields have been skipped. R6 would be ready to read or write the field numbered 6. In other words, moving to the tenth field or the sixth field would mean that ten fields or six fields would be skipped, including the zero field, which is the current field.

Try the following sample programs.

problem 4: Create a program that will read the information written to the file in problem #1.

program 4:
(C2P4)

```
10   D$ = CHR$(4)
20   PRINT D$;"OPEN FILE1"
30   PRINT D$;"READ FILE1"
40   INPUT A$,B$
50   PRINT A$
55   PRINT B$
60   PRINT D$;"CLOSE FILE1"
70   END
```

program description:

Line 30 -- prepares the file to be read from.
Line 40 -- reads the first two fields from the file.

The run of the above program should print JANUARY and FEBRUARY on the screen.

problem 5: Create a program that will read the 24 fields from the file created in problem #2.

program 5:
(C2P5)

```
10  D$ = CHR$ (4)
20  PRINT D$;"OPEN FILE2"
30  PRINT D$;"READ FILE2"
40  FOR X = 1 TO 12
50  INPUT A$,A
60  PRINT A$,A
70  NEXT X
80  PRINT D$;"CLOSE FILE2"
90  END
```

program description: Line 30 — prepares the file to be read from.
Line 50 — reads two fields from the file.

The run of the above program will print twelve names and numbers in columns on the screen.

problem 6: Create a program that will read the information written to the file in program 3 (C2P3).

program 6:
(C2P6)

```
10  D$ = CHR$(4)
20  PRINT D$;"OPEN FILE3"
30  FOR X = 1 TO 7
40  PRINT "READ FIELD #";X - 1
50  PRINT D$;"READ FILE3"
60  INPUT A$
70  PRINT A$
80  NEXT X
90  PRINT D$;"CLOSE FILE3"
100  END
```

program description: Line 50 — prepares the file to be read from.
Line 60 — reads a field from the file.

The run of the program will print the days of the week on the screen.

problem 7: Create a program that will write a file with the names of the twelve months of the year. Position the file so it will read the last six names and print them on the screen.

program 7:
(C2P7)

```
10  D$ = CHR$(4)
20  PRINT D$;"OPEN FILE30"
30  PRINT D$;"WRITE FILE30"
40  FOR X = 1 TO 12
50  READ A$
60  PRINT A$
70  NEXT X
80  PRINT D$;"CLOSE FILE30"
90  PRINT D$;"OPEN FILE30"
100  PRINT D$;"POSITION FILE30,R6"
110  PRINT D$;"READ FILE30"
120  INPUT A$
130  PRINT A$
140  IF A$ = "DECEMBER" THEN 160
150  GOTO 120
160  PRINT D$;"CLOSE FILE30"
170  DATA "JANUARY","FEBRUARY","MARCH","APRIL",
      "MAY","JUNE","JULY","AUGUST","SEPTEMBER",
      "OCTOBER","NOVEMBER","DECEMBER"
180  END
```

program description: Line 100 prepares the file at the field numbered 6. It skips over the six fields numbered 0 through 5.

problem 8: Create a program that will read the first three months and the last three months in problem 7 and print them out.

program 8:
(C2P8)

```
10  D$ = CHR$ (4)
20  PRINT D$;"OPEN FILE30"
30  PRINT D$;"READ FILE30"
40  INPUT A$
50  PRINT A$
60  C = C + 1
70  IF C = 3 THEN 90
80  GOTO 40
90  PRINT D$;"POSITION FILE30,R6"
100  PRINT D$;"READ FILE30"
110  C = 0
120  INPUT A$
130  PRINT A$
140  C = C + 1
150  IF C = 3 THEN 170
160  GOTO 120
170  PRINT D$;"CLOSE FILE30"
180  END
```

program description: Line 30 prepares the file for reading from.
Line 90 skips six fields from the current position so the file will be at the field numbered 10.

EXERCISE SET 2B

1. What file commands are necessary to read information from a file?

2. Write a program that will read a file named SAMPLEC2A3 with the following information:

field 0	-	South Dakota
1	-	Pierre
2	-	Minnesota
3	-	St. Paul
4	-	North Dakota
5	-	Bismarck
6	-	Montana
7	-	Helena

3. Correct the one error in the following program:

```
10  D$ = CHR$(4)
20  PRINT D$;"OPEN SAMPLE"
30  PRINT D$;"READ SAMPLE"
40  PRINT "HOW MANY FIELDS WILL BE READ"
50  INPUT N
60  FOR X = 1 TO N
70  INPUT A
80  PRINT A
90  NEXT X
100  PRINT D$;"CLOSE SAMPLE"
120  END
```

4. What is the READ statement for?

5. Write a program that will read the employee list named SAMPLEC2A2 used in problem 8 in Exercise 2A. The number of employees to be read should be requested at the start of the program. Use ten or fewer employees. Print the list on the screen.

6. What is the purpose of the POSITION statement?

7. What field would the following statement start reading at?

```
20  PRINT D$;"POSITION FILE18,R11"
```

8. Write a program that will read a file named SAMPLEC2A5, starting with the name HARRY WATSON and reading to the end. The file consists of the following names. Print the names on the screen.

JERRY JONES
SID WALKER
LANNY CARLSON
JOANN REID
HARRY WATSON
TOM STANTON
CARRIE HEIN

9. What four file commands must be used to position to a given field and to read to the end of the file?

10. Write a program that will change the last four names in the file named SAMPLE C2A5 in problem 8 to the following:

    JAMES ANDERSON
    TERRI LAWRENCE
    MARIE RUETER
    CANDACE JAW

2.4 CORRECTING AND ADDING TO A SEQUENTIAL FILE

Correcting a File

To make changes to a sequential file the contents of the file must first be read into an array. The array is then checked until the field is found that needs to be changed. The change or changes are then made to the array and the corrected array is written back to the file.

Try the following sample program.

problem 9: Create a program that will change the fifth field from JOHN DOE to JANE DOE. The file is called SAMPLEC2P9 and has ten fields.

program 9:
(C2P9)

```
10  D$ = CHR$ (4)
20  PRINT D$;"OPEN SAMPLEC2P9"
30  PRINT D$;"READ SAMPLEC2P9"
40  FOR X = 1 TO 10
50  INPUT A$(X)
60  NEXT X
70  FOR X = 1 TO 10
80  IF A$(X) = "JOHN DOE" THEN A$(X) = "JANE DOE"
90  NEXT X
100  PRINT D$;"CLOSE SAMPLEC2P9"
110  PRINT D$;"DELETE SAMPLEC2P9"
120  PRINT D$;"OPEN SAMPLEC2P9"
130  PRINT D$;"WRITE SAMPLEC2P9"
140  FOR X = 1 TO 10
150  PRINT A$(X)
160  NEXT X
170  PRINT D$;"CLOSE SAMPLEC2P9"
180  END
```

program description:
line 50 reads in the array from the file.
Line 80 checks to see what line has to be changed.
Line 100 closes the file so it can be reopened.
Line 110 the file is deleted to eliminate any possibility of extra bytes being left at the end of the file after rewriting.
Line 150 writes the corrected array back to the file.

In the above program the changed name, JANE DOE, was the same length as the original name, JOHN DOE, so the length of the file isn't changed. If thechanges produce a longer file, it still doesn't make any difference, because the entire file—not just a single field—is rewritten. The DELETE statement at line 110 is needed in case the changes produce a shorter file. When a shorter file is produced, a small number of bytes of data from the original file will be left attached to the end of the file. These extra bytes can cause problems when more information is appended to the file. The APPEND command is covered in the next section.

Adding to a File

You can add to the end of a sequential file by using the following file commands:

```
APPEND
WRITE
CLOSE
```

The WRITE and CLOSE statements were discussed in a previous section.

APPEND Statement

general form: ln PRINT D$;"APPEND name"

where: ln is any legal line number.
D$ is a control-D.
name is the file name.

example:

```
20  PRINT D$;"APPEND FILE19"
30  PRINT D$;"APPEND FILE20"
```

Try the following programs:

problem 10: Create a program that will write the rest of the months to the file written in problem 1.

program 10:
(C2P10)

```
10  D$ = CHR$(4)
20  PRINT D$;"APPEND FILE1"
30  PRINT D$;"WRITE FILE1"
40  READ A$
50  PRINT A$
60  IF A$ = "DECEMBER" THEN 80
70  GOTO 40
80  PRINT D$;"CLOSE FILE1"
90  DATA MARCH, APRIL, MAY, JUNE, JULY, AUGUST
100  DATA SEPTEMBER, OCTOBER, NOVEMBER, DECEMBER
110  END
```

program description: Line 20 — opens the file and positions the file at the end of the information already written on it.
Line 50 — adds to the file.

problem 11: Create a program that will either create or add to a file with the name of SAMPLE6.

program 11:
(C2P11)

```
10  D$ = CHR$ (4)
20  C = 0:D = 0
30  PRINT "ARE YOU CREATING OR ADDING TO A FILE."
40  PRINT "1=CREATE, 2=ADD"
50  INPUT N
60  IF N = 1 THEN 100
70  IF N = 2 THEN 220
80  PRINT "PLEASE TYPE IN 1 OR 2"
90  GOTO 50
100  PRINT D$;"OPEN SAMPLE6"
110  PRINT "HOW MANY FIELDS WILL BE ENTERED"
120  INPUT F
130  PRINT "ENTER ITEM NUMBER ";C + 1
140  INPUT A$
160  C = C + 1
170  PRINT D$;"WRITE SAMPLE6"
180  PRINT A$
190  PRINT D$
200  IF C = F THEN 350
210  GOTO 130
220  PRINT D$;"APPEND SAMPLE6"
230  PRINT "HOW MANY FIELDS WILL BE ADDED";
240  INPUT B
250  PRINT "ENTER ITEM NUMBER ";D + 1;" TO BE ADDED"
270  INPUT A$
290  D = D + 1
300  PRINT D$;"WRITE SAMPLE6"
310  PRINT A$
320  PRINT D$
330  IF D = B THEN 350
340  GOTO 250
350  PRINT D$;"CLOSE SAMPLE6"
360  END
```

program description:
Line 50 gives the user the options of creating or adding to a file.
Line 100 starts the creating option.
Line 220 starts the adding option.

SPECIAL FEATURE

When using sequential files, the file is read until a carriage return is noted at the end of the field. The only other delimiter is the comma. If a comma is used between two strings within a field, then the first string will be read into the first variable, the second string into the second variable, and so on. If only the first variable is read, then the message 'EXCESS IGNORED' will be printed. This process could be used if you want to use just part of the field. In the following example only the second names were to be printed instead of the entire name.

problem 12: Create a program that will create fields with first and second names of employees separated in the file with a comma. Read the names back and print only the second names on the screen.

program 12:
(C2P12)

```
10  D$ = CHR$(4)
20  PRINT D$;"OPEN FILE36"
30  PRINT D$;"WRITE FILE36"
40  PRINT "JOHN,JONES"
50  PRINT "TOM,LARSON"
60  PRINT D$;"CLOSE FILE36"
70  PRINT  D$;"OPEN FILE36"
80  PRINT D$"READ FILE36"
90  INPUT A$,B$
100  PRINT B$
110  IF B$ = "LARSON" THEN 130
120  GOTO 90
130  PRINT D$;"CLOSE FILE36"
140  END
```

program description:

Line 40 and 50 -- writes two fields ("JOHN, JONES" and "TOM, LARSON") to the file.

Line 90 -- reads one field into two variables. Since the field has a comma in it, the field must be read into two variables.

Line 100 -- prints on the screen only the second part of the field.

EXERCISE SET 2C

1. Write a program that will search for the name JOHN LARSON in the sequential file SAMPLEC2A7 on your diskette. When it is found, change it to JOHN LARSEN. There are eight fields in the file.

2. What four file commands are needed to correct a file?

3. What is the purpose of the APPEND statement?

4. Write a program that will add the names of the seven days of the week to a file called SAMPLEC2A8 that has other data.

5. There is one error in the following program. Can you find it?

```
10  D$ = CHR$(4)
20  PRINT D$;"APPEND SAMPLE2"
30  FOR X = 1 TO 4
40  INPUT A$
50  PRINT D$;"WRITE SAMPLE2"
60  PRINT A$
70  NEXT X
80  PRINT D$;"CLOSE SAMPLE2'
90  END
```

Review - Chapter 2

The following sequential file commands were discussed:

APPEND Statement

general form: LN PRINT D$;"APPEND name"

example: 50 PRINT D$;"APPEND FILE6"

purpose: Opens the file and positions it at the end of the file so that you can add information starting in the next field.

MON Statement

general form: ln PRINT D$;"MON C,I,O"

example: 42 PRINT D$;"MON C,I,O"

purpose: Monitors file commands, information taken from the file, and information sent to the file.

NOMON Statement

general form: ln PRINT D$;"NOMON C,I,O"

example: 13 PRINT D$;"NOMON C,I,O"

purpose: Turns off the monitor set by the MON statement.

OPEN Statement

general form: ln PRINT D$;"OPEN name"

example: 20 PRINT D$;"OPEN FILE1"

purpose: Sets the workspace to create a new file or to access an old file.

POSITION Statement

general form: ln PRINT D$;"POSITION name,Rp"

example: 45 PRINT D$;"POSITION FILE8,R3"

purpose: Prepares the file at the p field. If p = 3 then it would skip 0, 1, and 2 and set the pointer at the fourth location.

READ Statement

general form: ln PRINT D$;"READ name"

example: 40 PRINT D$;"READ FILE20"

purpose: Prepares the file so that it can be read.

WRITE Statement

general form: ln PRINT D$;"WRITE name"

example: 42 PRINT D$;"WRITE FILE30"

purpose: Prepares the file so that it can be written on.

SAMPLE PROGRAM

PROBLEM C2P13

Below is an example of a program using files. This program is designed to show the user how a final program should look.

The following program will allow the user either to create, add to, or list a sequential data file on the Apple II computer.

```
10  TEXT : HOME
20  D$ = CHR$ (4)
30  C = 0
40  VTAB 10: PRINT "THIS PROGRAM WILL ALLOW YOU TO CREATE"
50  PRINT : PRINT "ADD TO OR LIST A SEQUENTIAL DATA FILE."
60  PRINT : PRINT : PRINT "WOULD YOU LIKE TO CREATE A FILE, ADD"
70  PRINT : PRINT "TO A FILE,  LIST A FILE,"
80  PRINT : PRINT "OR END THE  PROGRAM?"
90  PRINT : PRINT "1=CREATE 2=ADD 3=LIST 4=END";
100 INPUT N
110 IF N < 1 OR N > 4 THEN 130
120 ON N GOTO 160,300,460,650
130 PRINT : PRINT "PLEASE TYPE IN A 1,2,3 OR 4."
140 GOTO 60
150 REM CREATE ROUTINE
160 PRINT : PRINT "NAME YOUR FILE";
170 INPUT A$
180 PRINT D$;"OPEN";A$
190 HOME
200 PRINT "HOW MANY FIELDS WILL BE ENTERED";
210 INPUT N
220 FOR X = 1 TO N
230 PRINT : PRINT "ENTER VALUE FOR FIELD #";X
240 INPUT B$
250 PRINT D$;"WRITE";A$
260 PRINT B$
270 PRINT D$
280 NEXT X
290 GOTO 590
300 REM ADD ROUTINE
310 PRINT : PRINT "NAME OF FILE";
320 INPUT A$
330 PRINT D$;"APPEND";A$
340 HOME
350 PRINT : PRINT "HOW MANY FIELDS WILL BE ADDED";
360 INPUT N
370 C = C + 1
380 PRINT : PRINT "ENTER ADDED VALUE #";C
390 INPUT B$
400 PRINT D$;"WRITE";A$
410 PRINT B$
420 PRINT D$
430 IF C = N THEN 590
440 GOTO 370
```

```
450 REM LIST ROUTINE
460 PRINT : PRINT "NAME OF FILE";
470 INPUT A$
480 PRINT D$;"OPEN";A$
490 HOME
500 PRINT "HOW MANY FIELDS IN YOUR FILE";
510 INPUT N
520 HOME
530 PRINT TAB ( 10);"CONTENTS OF THE FILE"
540 PRINT D$;"READ";A$
550 INPUT B$
560 PRINT : HTAB 15: PRINT B$
570 C = C + 1
580 IF C<>N THEN 540
590 PRINT D$;"CLOSE";A$
600 PRINT : HTAB 6: INVERSE
610 PRINT "YOUR FILE IS COMPLETE"
620 NORMAL
630 PRINT : PRINT "PRESS SPACE BAR TO CONTINUE";
640 GET A$: PRINT :  GOTO 10
650 END
```

Review Quiz - Chapter 2

1. Name the three file statements used when reading an entire sequential file.

2. Name the four file statements used when reading the last half of a sequential file.

3. Name the three file statements used when adding to the end of a file.

4. Name the three file statements used when writing to a file.

5. When using the statement 15 PRINT D$;"MON,C,O" what will be printed for the following program?

```
10  D$ = CHR$(4)
20  PRINT D$;"OPEN FILE8"
30  PRINT D$;"WRITE FILE8"
40  READ A$
50  PRINT A$
60  IF A$ = "GHOST" THEN 80
70  GOTO 40
80  PRINT D$;"CLOSE FILE8"
90  DATA HOUSE, HALLOWEEN, MASK, GHOST
100  END
```

6. Write a program that will read twenty names of people, numbers, and prices from a sequential file named SAMPLEC2A9. The three items are in the same field, separated by commas. Print all three items on the screen for all twenty groups.

7. Write a program that will read only the last ten groups of three items from the file in problem 6. Print all three items on the screen for the last ten groups.

8. Write a program that will correct a name in the file used in problem 6. The name should be 'BENJAMIN MOORE' instead of 'BENJAMIN MORE'.

9. Correct the following program:

```
10  D$ = CHR$(4)
20  PRINT D$;"OPEN FILE1"
30  PRINT D$;"INPUT FILE1"
40  INPUT A$
50  PRINT D$;"CLOSE FILE1"
60  END
```

10. Correct the following program:

```
10  D$ = CHR$(4)
20  PRINT D$;"OPEN FILE2"
30  PRINT D$;"WRITE FILE2"
40  WRITE A$
50  PRINT D$;"CLOSE FILE2"
60  END
```

CHAPTER 3

RANDOM ACCESS FILES

3.1 INITIAL REMARKS

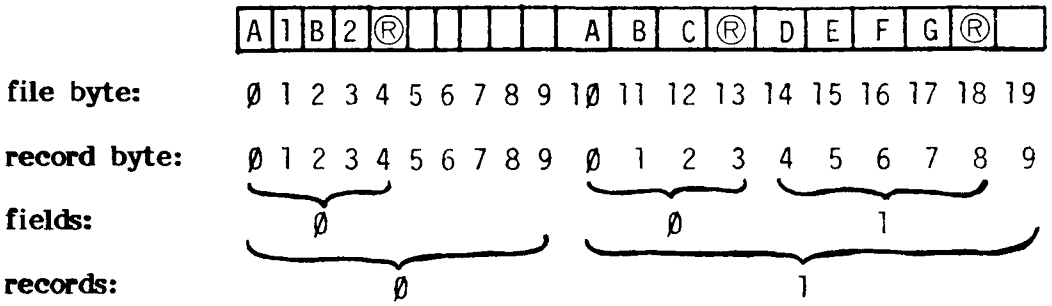
Random access files should be used when the information in the file will be changed often, when not all the data is needed every time the file is used, and when quick access to a single item of data is important.

Careful planning must take place before using a random access file. The exact information to be stored on the file, how it will be stored, and how it will be recalled from the file must be determined before using a random access file. Determining the length of the records in the file is the next step to using the file. The record length is determined by finding the maximum number of bytes or characters that will be in any one of the records. Each letter, digit, special symbol, and carriage return must be counted to find the record length. The records always take up the number of characters specified, even if there are fewer characters written to the record. Good planning will save much space on the diskette.

When writing to a record in a random access file, care must be taken so more characters are not written to a record than is specified by its length. If too many characters are written to a record, the carriage return at the end of the last field in the record is lost and the extra characters will be written to the next record. If the field without a carriage return is read, the computer does not know where to stop reading and will read all or part of the next record.

The length of the records in a random access file must be remembered because the length cannot be determined at a later time. A way to remember the record lengths of a random access file is to make the record length part of the name of the file. For example, if a file has record lengths of 150, the name of the file might be ABC150. The last three characters in the file name is the length of the records in the file.

Below is a diagram of a random access file with ten bytes allowed in each record. File bytes, record bytes, fields, and records all start numbering from zero.


A 1 B 2 ® A B C ® D E F G ®
file byte:
Ø 1 2 3 4 5 6 7 8 9 1Ø 11 12 13 14 15 16 17 18 19
record byte:
Ø 1 2 3 4 5 6 7 8 9 Ø 1 2 3 4 5 6 7 8 9
fields:
Ø Ø 1
records:
Ø 1

Each record in this file can contain up to ten bytes or characters of information. The maximum record length allowed in any file is 32,767 bytes. A character can be any number, letter, special symbol, or carriage return. A record can contain as many fields as the length of the record will allow. The carriage return is the terminating character for each field in the record. Since each record in a random access file can have more than one field, each record can be thought of as a small sequential file.

3.2 CREATING A RANDOM ACCESS FILE

The file commands needed to enter data to a random file on a diskette are:

```
OPEN
WRITE
CLOSE
```

OPEN Statement

general form:

```
ln  PRINT D$;"OPEN name,Lb"
```

where:

ln is any legal line number.
D$ is a control-D.
name is the name of a file.
b is the number of bytes in each record.

example:

```
100  PRINT D$;"OPEN ALPHA,L200"
132  PRINT D$;"OPEN BETA,L23"
281  PRINT D$;"OPEN ";F$;",L";B
```

where F$ and B are variables for the file name and record length. These variables must be set by the program before the OPEN statement.

purpose:

When the OPEN statement is executed, the computer sets the length of the records and reserves a 595-byte file buffer in the computer. Information is stored in the buffer before it is written to the diskette.

Alternate Forms of the OPEN Statement

general form:

```
ln  PRINT D$;"OPEN name,Lb,Ss"
ln  PRINT D$;"OPEN name,Lb,Dd"
ln  PRINT D$;"OPEN name,Lb,Ss,Dd"
```

where:

s is the disk drive controller slot number. The number can be from 1 to 7.
d is the disk drive number. The number can be 1 or 2.

example:

```
100  PRINT D$;"OPEN ALPHA,L2000,S4"
132  PRINT D$;"OPEN BETA,L23,D2"
281  PRINT D$;"OPEN FILE5,L60,S7,D1"
```

purpose:

An alternate form of the OPEN statement is needed when more than one disk drive is connected to the computer. If two disk drives are connected to the same controller card, the Dd form of the statement would be used to go between disk drive one, D1, and disk drive two, D2. If the two disk drives each use a controller card, the Ss form of the OPEN statement will be used. If more than two disk drives are used, the final form will have to be used to get to the correct controller and disk drive.

WRITE Statement

general form:

```
ln   PRINT D$;"WRITE name,Rx"
```

where:

ln is any legal line number.
D$ is a control-D.
name is a file name.
x is the number of the record to be written.

example:

```
110  PRINT D$;"WRITE ALPHA,R1"
134  PRINT D$;"WRITE BETA,R22"
282  PRINT D$;"WRITE ";F$;",R";E
```

where F$ and E are variables for the file name and record number. They must be set before the statement is executed.

purpose:

The WRITE statement positions the file to the record to be written. Any PRINT statement following a WRITE statement will be printed on the file at the record indicated and not on the screen. Care must be taken to terminate printing to the file before asking for input or the question mark from the input will go to the file. The shortest way to stop printing to the file is to use the statement PRINT D$.

Note: Error messages and other unwanted characters can also get written to the file.

Alternate Form of the WRITE Statement

general form:

```
ln   PRINT D$;"WRITE name,Rx,By"
```

where:

ln is any legal line number.
D$ is a control-D.
name is a file name.
x is the number of the record.
y is the yth byte in the record.

example:

```
120  PRINT D$;"WRITE ALPHA,R1,B5"
283  PRINT D$;"WRITE ";F$;",R";X;",B";Y
```

purpose:

For the very expert, knowledgeable, and experienced programmer. When the By form of the WRITE statement is used, writing will start at the yth byte of the record.

CLOSE Statement

The CLOSE statement is very important. If a CLOSE statement is not used, the data from any OPEN file could be lost or bad data could be added to the file.

If a program is stopped abnormally and files have been used in the program, it is a good idea to type CLOSE before going on. This makes sure all files are closed.

general form:

```
ln  PRINT D$;"CLOSE name"
```

where:

ln is any legal line number
D$ is a control-D.
name is a file name.

example:

```
120  PRINT D$;"CLOSE ALPHA"
283  PRINT D$;"CLOSE ";F$
```

where F$ is a variable for the file name.

purpose: The CLOSE statement writes all remaining characters in the file buffer to the file, releases the file from the program, and releases the buffer associated with the file.

Alternate Form of the CLOSE Statement

general form:

```
ln  PRINT D$;"CLOSE"
```

purpose: This statement should be used near the end of a program to make sure all files are closed.

Try the following sample programs:

problem 1: Create a random access file that has the words "one," "two," "three," "four," and "five" in records 1, 2, 3, 4, and 5, respectively.

program 1:
(C3P1)

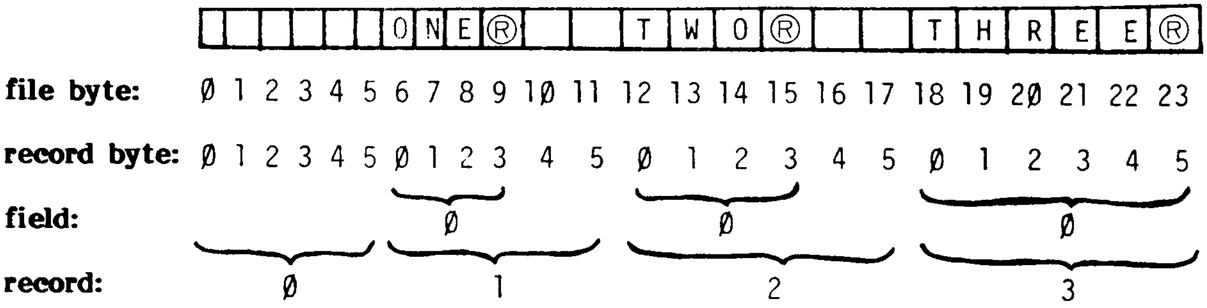
```
10  D$ = CHR$(4)
20  PRINT "STARTING THE PROGRAM"
30  PRINT D$;"OPEN RFONE006,L6"
40  FOR I = 1 TO 5
50  READ A$
60  PRINT D$;"WRITE RFONE006,R";I
70  PRINT A$
```

```
80    NEXT I
90    PRINT D$
100   DATA "ONE","TWO","THREE","FOUR","FIVE"
110   PRINT "THE RANDOM FILE RFONE006 HAS BEEN"
115   PRINT "CREATED"
120   PRINT D$;"CLOSE RFONE006"
130   END
```

program description:

Line 20 -- opens a file called RFONE006 with records six bytes long.
Line 50 -- reads one item of data from line 100.
Line 60 — indicates which record in RFONE006 will be written to.
Line 70 -- the variable A$ is printed to the file.
Line 90 — stops writing to the file.
Line 100 -- data used.
Line 120 — closes file RFONE006.

After running this program, the file will show up on the catalog of the diskette. A diagram of the first four records of RFONE006 is shown below.


O N E ®
T W O ®
T H R E E ®
file byte:
Ø 1 2 3 4 5 6 7 8 9 1Ø 11 12 13 14 15 16 17 18 19 2Ø 21 22 23
record byte:
Ø 1 2 3 4 5 Ø 1 2 3 4 5 Ø 1 2 3 4 5 Ø 1 2 3 4 5
field:
Ø
Ø
Ø
record:
Ø
1
2
3

problem 2: Write a program that will create a random file with information entered from the keyboard. Have each record ten bytes long and write entry one to record one, entry two to record two, and so on.

program 2:
(C3P2)

```
10    D$ = CHR$(4)
20    PRINT "PROGRAM TO CREATE RFTWO010"
30    PRINT D$;"OPEN RFTWO010,L10"
40    PRINT "ENTER WORDS OF LESS THAN 10 CHARACTERS"
50    FOR I = 1 TO 3
60    PRINT "ENTER WORD ";I;"  ";
70    INPUT A$
80    IF LEN(A$) > 9 THEN 60
90    PRINT D$;"WRITE RFTWO010,R";I
100   PRINT A$
110   PRINT D$
120   NEXT I
130   PRINT "DONE"
140   PRINT D$;"CLOSE RFTWO010"
150   END
```

The difference between program 2 and program 1 is slight but important. Statement 110 PRINT D$ is inside the loop in program 2, while in program 1 it is outside the loop. In program 2 it has to be inside the loop to stop printing to the file or statement 60 PRINT "ENTER WORD ";I; " "; would be printed to the file. Be very careful when creating a program similar to this one. Make sure the statement PRINT D$ is in the right place in the program.

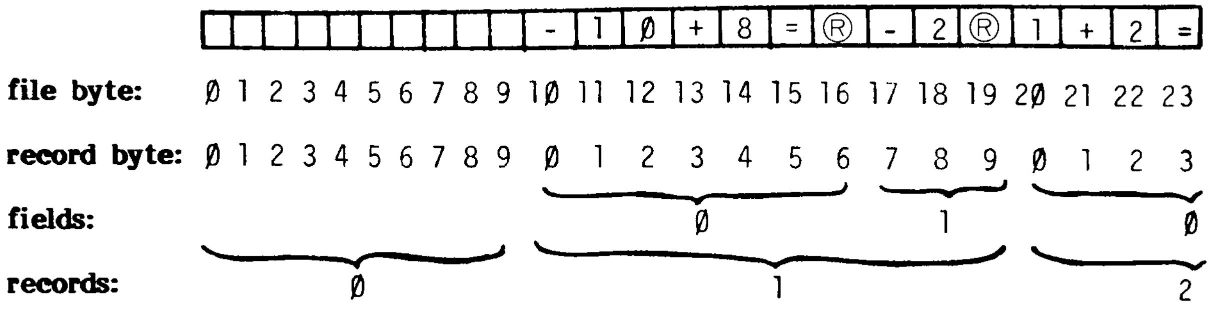
problem 3: Write a random file that has two fields per record.

program 3:
(C3P3)

```
10    D$ = CHR$(4)
20    PRINT "PROGRAM TO CREATE RANDOM FILE"
30    PRINT D$;"OPEN RFTHREE010,L10"
40    FOR I = 1 TO 3
50    READ A$,B$
60    PRINT D$;"WRITE RFTHREE010,R";I
70    PRINT A$
80    PRINT B$
90    NEXT I
100   PRINT D$
110   DATA "-10+8 =","-2","1+2 =","3","8+10 =","18"
120   PRINT "THE FILE HAS BEEN CREATED"
130   PRINT D$;"CLOSE RFTHREE010"
140   END
```

Program 3 is similar to program 1 except that it reads two data items at a time and prints both to the same record in the file. This will cause each record to have two fields.

Below is a diagram of the file.


- 1 Ø + 8 = Ⓡ - 2 Ⓡ 1 + 2 =
file byte: Ø 1 2 3 4 5 6 7 8 9 1Ø 11 12 13 14 15 16 17 18 19 2Ø 21 22 23
record byte: Ø 1 2 3 4 5 6 7 8 9 Ø 1 2 3 4 5 6 7 8 9 Ø 1 2 3
fields: Ø 1 Ø
records: Ø 1 2

Statements 70 PRINT A$ and 80 PRINT B$ were written as two statements instead of one so that a carriage return would be placed between A$ and B$ in the file. If the statement were written as PRINT A$,B$ no carriage return would have been placed between them in the file and the two variables would act as one variable when being read from the file. There are other ways to separate variables in a file, but they will not be discussed here.

Special Statements

It is impossible to tell what is being written to the file in the above program. If there were an error in the program, the error message would have been written to the file and not the screen. It may even look as if there were no errors at all. Because of this there are the following two special statements.

MON Statement

general form:	PRINT D$;"MON,C,I,O"
where:	C — displays disk commands like OPEN, WRITE, and CLOSE. I -- displays the information that is read from the file. O — displays the information that is written to the file. The three parameters--C, I, and O--can be used in any combination or order.
purpose:	To monitor disk activities such as when files are opened and closed and what is written to and read from files.

NOMON Statement

When the MON statement is used the monitoring of disk activities will continue from program to program until a NOMON statement is executed.

general form: PRINT D$;"NOMON,C,I,O"

where: C — turns off monitoring disk commands.
I -- turns off monitoring items being input from a file.
O — turns off monitoring items being output to a file.

purpose: To turn off the MON statement and stop printing of file commands and information taken from and sent to a file. If the MON statement is not turned off it carries over to the next program or command.

Rerun the program above with the following statements added:

```
15    PRINT D$;"MON,C"
135   PRINT D$;"NOMON,C"
```

Rerun again with:

```
15    PRINT D$;"MON,O"
135   PRINT D$;"NOMON,O"
```

Rerun again with:

```
15    PRINT D$;"MON,C,O"
135   PRINT D$;"NOMON,C,O"
```

The I parameter has no meaning with this program because there is no input from the diskette.

EXERCISE SET 3A

1. What does "random access" mean?

2. What is the first record number in a random access file?

3. What is the shortest record length that is logically usable in a random access file?

4. Which of the following statements are correctly written for random access files? For these examples D$ is a control-D.

   a. PRINT D$;"OPEN FILE"
   b. PRINT CHR$(4);"CLOSE FILE"
   c. PRINT D$;"WRITE FILE,RO"
   d. PRINT CHR$(4);"OPEN";F$;"L100"
   e. PRINT D$;"CLOSE"

5. Why is the CLOSE statement necessary?

6. Write a program to print the first 25 prime numbers to a file. Put the first prime number in record one, the second in record two, and so forth.

7. Number each person in the computer class sequentially. Write the number of people in the class to record zero in the file and then write each person's name and telephone number to the same record as the person's number.

8. Write a program that will allow a person to enter a file name and enter 25 spelling words. Have the words saved on the file, one word per record, starting with record one.

9. Rewrite the program in problem 8 and allow room on each record for: (1) the number of times a word was used; (2) the number of times it was spelled correctly; and (3) the percentage of times it was spelled correctly.

10. Create a random file with the names of the students in the programming class. Have the first and last names of each student in separate fields within the same record. Start the second field at byte 25 in the record.

3.3 READING A RANDOM DATA FILE

The file commands needed to read a random access file are:

```
OPEN
READ
CLOSE
```

The OPEN and CLOSE statements were discussed in Section 3.2. The statements are the same for this section.

READ Statement

general form:

```
ln   PRINT D$;"READ name,Rx"
```

where:

ln is any legal line number.
D$ is a control-D.
name is a file name.
x is the record number to be read.

example:

```
130  PRINT D$;"READ AHPLA,R5"
136  PRINT D$;"READ ATEB,R0"
284  PRINT D$;"READ";F$;",R";E
```

where F$ is a file name and E is the record number to be read.

purpose:

The READ statement positions the file to the record to be read. Any INPUT or GET statement following a READ statement causes characters to be read from the file. Out-of-data error messages will result if too much input is requested from one record. The PRINT D$ statement will terminate reading from a file.

Alternate Forms of the READ Statement

general form:

```
ln   PRINT D$;"READ name,Rx,By"
```

where:

ln is any legal line number.
D$ is a control-D.
name is a file name.
x is the number of the record to be read.
y is the byte in the record where reading is to start.

example:

```
130  PRINT D$;"READ ALPHA,R5,B10"
136  PRINT D$;"READ";F$;",R";E;",B";T
```

purpose:

For experienced programmers. Sets the file to the byte in the record where reading is to start.

Try the following programs.

problem 4: Read and print the first, third, and fifth records in the file created by program 1.

program 4:
(C3P4)

```
10   D$ = CHR$(4)
20   PRINT D$;"OPEN RFONE006,L6"
30   I = 1
40   PRINT D$;"READ RFONE006,R";I
50   INPUT A$
60   PRINT D$
70   PRINT A$
80   I = I + 2
90   IF I < 7 THEN 40
100  PRINT D$;"CLOSE RFONE006"
110  END
```

If program 1 and program 4 are entered and executed correctly, the output will be:

```
ONE
THREE
FIVE
```

problem 5: Randomly read two records from the file created by program 3. Ask for the answer to the question and check for the correct answer.

program 5:
(C3P5)

```
10   D$ = CHR$(4)
20   PRINT D$;"OPEN RFTHREE010,L10"
30   FOR I = 1 TO 2
40   J = INT(RND(1) * 3) + 1
50   PRINT D$;"READ RFTHREE010,R";J
60   INPUT Q$,A$
70   PRINT D$
80   PRINT "WHAT IS ";Q$;
90   INPUT B$
100  IF B$ = A$ THEN PRINT "CORRECT"
110   IF B$      A$ THEN PRINT "WRONG THE ANSWER
      IS ";A$
120  NEXT I
130  PRINT D$;"CLOSE RFTHREE010"
140  END
```

program description:

Line 40 -- the random number J is generated.
Line 50 -- the file RFTHREE010 is set so record J can be read.
Line 60 -- the question and answer is read from record J.
Line 80 -- the question is asked.
Line 90 -- the answer is entered.
Line 100 and 110 -- checks to see if the answer is correct.

Program 3 in Section 3.2 and program 5 above show how to create a random file of questions and answers and then have them randomly asked.

EXERCISE SET 3B

1. What does the READ statement do?

2. What is the general form of the alternate READ statement?

3. Which of the following statements are correct forms of the random access file READ statement?

   a. PRINT D$;"READ ABC,L6"
   b. PRINT D$;"READ ";F$;",R3"
   c. PRINT "READ XYZ,RO"
   d. PRINT CHR$(4);"READ QA,R";P
   e. PRINT CHR$(4);"READ GH023"

4. Using the random file created in question 6 of Exercise Set 3A, print the sixth, twelfth, and eighteenth prime numbers.

5. Write a program that will verify that the file created in question 7 of Exercise Set 3A is correct.

6. Write a program to use the file created in problem 9 of Exercise Set 3A. Have the program randomly select spelling words and keep track of how many times each was used and spelled correctly, and the percentage of times each was spelled correctly. Write the information back to the file.

7. Write a program that will read and print only the last names from the file created in problem 10 of Exercise Set 3A.

3.4 CORRECTING AND ADDING TO A RANDOM ACCESS FILE

Correcting a File

To change data in a random file, the record to be changed, the number of fields in the record, and the number of characters in the fields are important to know.

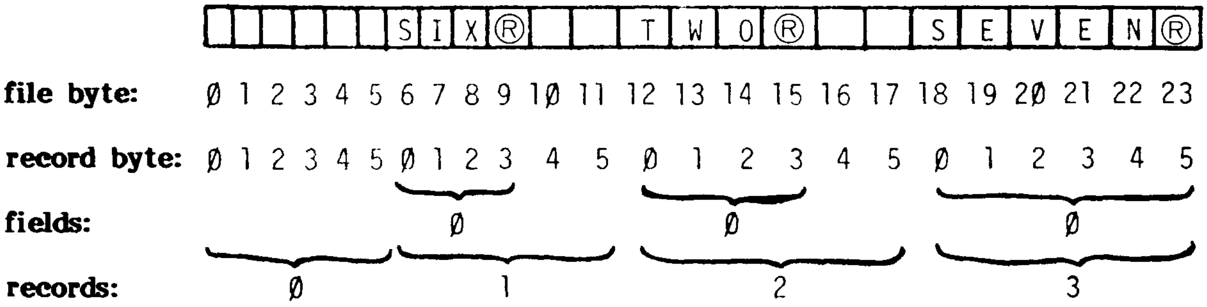
Try the following program:

problem 6: Using the file from problem 1, change record 1 to six and record 3 to seven.

program 6:
(C3P6)

```
10 D$ = CHR$(4)
20 PRINT "STARTING PROGRAM"
30 PRINT D$;"OPEN RFONE006,L6"
40 PRINT D$;"WRITE RFONE006,R1"
50 PRINT "SIX"
60 PRINT D$
70 PRINT "PRINTING NOW AT RECORD 3"
80 PRINT D$;"WRITE RFONE006,R3"
90 PRINT "SEVEN"
100 PRINT D$
110 PRINT "FINISHED - CLOSING FILE"
120 PRINT D$;"CLOSE RFONE006"
130 END
```

A diagram of what RFONE006 now looks like is shown below:


S I X ® T W O ® S E V E N ®
file byte:
Ø 1 2 3 4 5 6 7 8 9 1Ø 11 12 13 14 15 16 17 18 19 2Ø 21 22 23
record byte:
Ø 1 2 3 4 5 Ø 1 2 3 4 5 Ø 1 2 3 4 5 Ø 1 2 3 4 5
fields:
Ø
Ø
Ø
records:
Ø
1
2
3

There were no problems in making the two changes because the new data had the same number of characters as the old data.

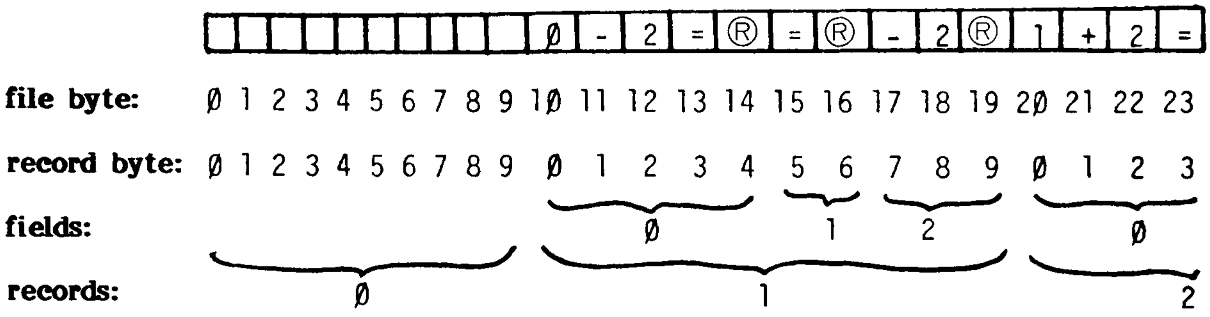
problem 7: Write a program that will do modifications to questions or answers in the file created by program 3. To test the program, change question 1 to "0-2=", but don't change its answer.

program 7:
(C3P7)

```
10 D$ = CHR$(4)
20 PRINT D$;"OPEN RFTHREE010,L10"
30 PRINT "WHICH PROBLEM DO YOU WANT TO CHANGE"
40 INPUT P
50 PRINT D$;"READ RFTHREE010,R";P
60 INPUT Q$,A$
70 PRINT D$
80 PRINT "PROBLEM IS ";Q$;" ANSWER IS ";A$
90 PRINT "WHAT IS THE NEW QUESTION";
100 INPUT Q$
110 PRINT "DO YOU WANT TO CHANGE THE ANSWER";
120 INPUT B$
130 IF LEFT$(B$,1) = "N" GOTO 160
140 PRINT "ENTER THE ANSWER";
150 INPUT A$
160 PRINT D$;"WRITE RFTHREE010,R";P
170 PRINT Q$
180 IF LEFT$(B$,1) = "Y" THEN PRINT A$
190 PRINT D$
200 PRINT D$;"CLOSE RFTHREE010"
210 END
```

When this program is run and problem 1, "-10+8= ", is changed to "0-2=", but the answer is not changed, the data file will have bad information in it.

A diagram of the new RFTHREE010 is shown below:


Ø - 2 = ® = ® - 2 ® 1 + 2 =
file byte: Ø 1 2 3 4 5 6 7 8 9 1Ø 11 12 13 14 15 16 17 18 19 2Ø 21 22 23
record byte: Ø 1 2 3 4 5 6 7 8 9 Ø 1 2 3 4 5 6 7 8 9 Ø 1 2 3
fields: Ø 1 2 Ø
records: Ø 1 2

Record 1 now has three fields. The first field is "0 - 2 = ", the second field is "=", and the third field is "-2". Since the second field is the answer to the questions, the answer to question 3 will be "=". This type of problem, where extra fields are being created in a record, will show up when fewer characters are rewritten to a record than were in the record to start with. If all fields are always rewritten to the record, more fields may be created, but the extra fields should not interfere with the program.

Adding to a Random Access File

There are no special tricks in adding to a random access file. The same program used to create a random file can be used to add to the file. Adding to a random file is done by setting the record on the WRITE statement to a record past the current end of the file.

problem 8: Add to the file created with problem 1.

problem 8:
(C3P8)

```
10 D$ = CHR$ (4)
20 PRINT D$;"OPEN RFONE006,L6"
30 PRINT "ENTER A STRING WITH LESS THAN"
40 PRINT "SIX CHARACTERS"
50 INPUT S$
60 IF LEN (S$) > 5 THEN 30
70 PRINT "WHICH RECORD SHOULD THIS GO TO";
80 INPUT A
90 PRINT D$;"WRITE RFONE006,R";A
100 PRINT S$
110 PRINT D$
120  PRINT D$;"CLOSE RFONE006"
130 END
```

The only problem with a program like this is that a record may get over-written and destroyed. A good way to keep track of the number of records that have been written to the file is to save the number of the last record written in record zero.

EXERCISE SET 3C

1. If both the question and answer were rewritten to record 3 in problem 7, how many fields would there be in record 3 and what would they be?

2. What would happen if more characters are rewritten to a record of a random access file?

3. Why is it important to know how many fields there are in each record?

4. How can the number of characters in a field be found?

5. What can be done to make sure more fields are not created in a record of a random file?

6. Write a program to make changes to any record in the file RFTWO010, created in program 2 in Section 3.2.

7. Write a program that will allow changes to the spelling words in the file created by problem 9 in Exercise Set 3A. Make sure to set the counters (times used, times correct, and percentage correct) back to zero.

8. Write a program that can check and correct the contents of the file created by problem 10 in Exercise Set 3A.

9. Use the program from problem 6, Exercise Set 3A and add the next ten prime numbers to the file.

10. Rewrite the program from problem 9, Exercise Set 3A so that words can be added to the file. In record zero keep track of the number of words in the file.

11. Write a program that will add names to the file created by the program in problem 7, Exercise Set 3A. What is used to know which record to starting writing to?

Review - Chapter 3

In Chapter 3, random files commands were introduced.

CLOSE Statement

general form: PRINT D$;"CLOSE A"

The CLOSE statement is needed to insure the safety of data on the file.

MON Statement

general form: PRINT D$;"MON,C,I,O"

The MON statement is needed to help show what file commands are being executed and to tell what is being written to and read from files.

NOMON Statement

general form: PRINT D$;"NOMON,C,I,O"

The NOMON statement is needed to turn off the MON statement.

OPEN Statement

general form: PRINT D$;"OPEN A,L60"

The L parameter must be in the statement. The OPEN statement is needed before a file can be read from or written to. The L indicates the length of the records in the file.

READ Statement

general form: PRINT D$;"READ A,R10"

The R parameter is optional, but needed most of the time. The READ statement is needed so that a file can be read from. The R indicates the record in the file to be read.

WRITE Statement

general form: PRINT D$;"WRITE A,R6"

The R parameter is optional, but needed most of the time. The WRITE statement is needed to set a file so that it can be written to. The R indicates the record in the file to be written to.

The problems and examples given should be studied, entered into the computer, and executed before proceeding to the review quiz.

SAMPLE PROGRAM

PROBLEM C3P9

Below is a sample program which demonstrates both reading and writing to a random file.

```
10  TEXT:HOME
20  D$=CHR$(4)
30  PRINT D$;"OPEN SAMPLEC3P9,L20,D1"
40  VTAB 3: PRINT "WHICH PROBLEM DO YOU WANT TO" : PRINT "CHANGE
    (1-3)";: CALL -958
50  INPUT P$
60  IF VAL (P$) < 1 OR VAL (P$) > 3 THEN 40
70  P =  (VAL (P$) - 1) * 2 + 1
80  PRINT D$;"READ SAMPLEC3P9,R";P
90  INPUT Q$,A$
100 PRINT D$
110 VTAB 6: PRINT "THE PROBLEM IS "; Q$ : PRINT "THE ANSWER IS";A$
120 VTAB 9: PRINT "WHAT IS THE NEW QUESTION";: INPUT Q$
130 VTAB 12: PRINT "DO YOU WANT TO CHANGE THE ANSWER ";: CALL -958
140 INPUT B$:B$ = LEFT$ (B$,1)
150 IF B$ <> "N" AND B$ <> "Y" THEN 130
160 IF B$ = "N" THEN 180
170 VTAB 15: PRINT "ENTER THE ANSWER ";: INPUT A$
180 PRINT D$;"WRITE SAMPLEC3P9,R";P
190 PRINT Q$
200 PRINT A$
210 PRINT D$
220 PRINT D$;"CLOSE SAMPLEC3P9"
230 END
```

Review Quiz - Chapter 3

1. Which of the following statements are correct for a random file OPEN statement?

a. PRINT D$;"OPEN F",L10

b. PRINT "D$;OPEN F,L10"

c. PRINT CHR$(10);"OPEN F,L10"

d. PRINT D$;"OPEN F,L10"

e. PRINT D$;"OPEN F"

2. Which of the following statements are correct for a random file WRITE statement?

a. PRINT CHR$(4);"WRITE F"

b. PRINT CHR$(4);"WRITE F,R";8

c. PRINT D$;"WRITE ";F$;"R60"

d. PRINT D$;"WRITE ";F$;",R6,B10"

e. PRINT "D$;WRITE ";F$,R12

3. Which of the following statements are correct for a random file CLOSE statement?

a. PRINT "CLOSE A"

b. PRINT D$;"CLOSE"

c. PRINT D$;"CLOSE,A"

d. PRINT CHR$(4);"CLOSE A"

e. PRINT CHR$(4); CLOSE;F$

4. Which of the following statements are written correctly for a random file READ statement?

a. PRINT D$;"READ F$"

b. PRINT CHR$(4);"READ F"R7

c. PRINT CHR$(4);"READ ";F$,R12

d. PRINT D$;"READ F,R";I

e. PRINT D$;"READ"F

5. Which of the following MON statements are written correctly for random files?

a. PRINT D$;"MON,C,I"

b. PRINT D$;"MON,ALL"

c. PRINT D$;"MON,INPUT"

d. PRINT D$;"MON,O"

e. PRINT D$;"MON,O,I,C"

6. Which of the following NOMON statements are written correctly for random files?

a. PRINT CHR$(4);"NOMON"

b. PRINT D$;"NOMON,I"

c. PRINT CHR$;"NOMON,C,O"

d. PRINT D$;"NOMON,O"

e. PRINT CHR$(4);"NOMON,C,I,O"

7. Write a program that will put student names, total points scored on tests, and total tests taken on a random access file. Number each student and put the data for each student on the corresponding record in a random file. To start, put zeros for total points and tests taken.

8. Write a program that will update the student data from problem 7. Be able to add a test score to the total points and increase the total tests taken.

9. Modify the program from problem 8 to keep track of total points scored and total tests taken by the class on record zero of the file.

10. Write a program that will allow students to get their individual data from the file used in problem 8. Also print out the students' average points per test and the class's average points per test.

CHAPTER 4

BINARY FILES

4.1 Initial Remarks

When you are preparing to work with binary data files, the data to be made into the file must be analyzed. The data must be placed into the file in an organized manner so that it can be retrieved with little effort. Before beginning you must remember that one character in a string will take up one byte of computer memory and that one byte of memory can only hold the numbers from 0 to 255. After analyzing your data you must determine the amount of memory the data will take up. When the amount has been determined, you must decide if and where the file will fit into the computer's memory.

4.2 CREATING A BINARY DATA FILE

One program statement and one file command are needed to create a data file. The statement is POKE and the file command is BSAVE.

POKE Statement

general form: ln POKE a,n

where: ln is any legal line number.
a is a legal memory address.
n is a number from 0 to 255.

example:

```
10   POKE 31500,16
20   POKE 768,123
```

purpose: To change the content of a memory location or to store data in a memory location. The memory locations can have an address from 0 to the maximum amount of memory in the computer. For a 16K Apple it is 16,384; for 32K it is 32,768, and for 48K it is 49,152. If data is POKEd into certain memory locations, the operations of the computer or computer program may be altered.

BSAVE Statement

general form: ln PRINT D$;"BSAVE name,Ax,Ly"

where: ln is any legal line number.
D$ is a control-D.
name is the binary file name.
x is the starting address of the file. If x is in hexadecimal, it must be preceded by a $.
y is the length of the file. If y is in hexadecimal, it must be preceded by a $.

example:

```
30   PRINT D$;"BSAVE AJT,A31000,L5400"
40   PRINT D$;"BSAVE NAUC, A$400,L8192"
50   PRINT D$;"BSAVE PDM,A$4000,L$2000"
```

purpose: To save the exact bit-by-bit information in a range of memory locations.

Try the following sample programs:

problem 1: Write a program that will save the following 25 numeric data items in a binary file. The data is 8, 92, 16, 84, 32, 168, 64, 200, 128, 0, 11, 251, 72, 181, 3, 99, 255, 54, 247, 108, 231, 216, 19, 66, 25.

problem discussion:

Each byte or memory location in the Apple computer can hold a number from 0 to 255. Since the 25 data items are all less than 255, there is no problem with too big of a number. If a 32K Apple is now being used, the best place to POKE data is above HGR and below DOS, between memory locatons 16385 and 22015. HIMEM must be set below the starting address of the binary data, or the data or program will be destroyed.

program 1:
(C4P1)

```
10    HIMEM:16400
20    D$ = CHR$(4)
30    FOR I = 1 TO 25
40    READ A
50    B = 16400 + I
60    POKE B,A
70    NEXT I
80    PRINT D$,"BSAVE BDF,A16401,L25"
90    DATA 8,92,16,84,32,168,64,200,128,0,11,251
100   DATA 72,181,3,99,255,54,247,108,231,216
110   DATA 19,66,25
120   END
```

program discussion:

Between memory locations 16385 and 22015 there are 5630 bytes of memory to store data. The program only needs 25 bytes, so it does not matter where between 16385 and 22015 it is POKEd.

Line 10 —— high memory is set to 16400.
Line 50 --- B is set to where the next data item will be POKEd.
Line 60 —— the data is POKEd into memory.
Line 80 --- the binary data file "BDF" is saved with starting
address 16401 and a length of 25.

program 2:

Rewrite program 1 to handle the following 25 data items. The data is 17.6, 12.4, 32, 9.2, 101.5, 66.1, 98.6, 200, 7.2, 12.8, 21.1, 43.7, 182, 0, 76.5, 57.1, 81.1, 18.8, 13.3, 12.2, 6.3, 8.2, 11.3, 182.6, 133.6.

problem discussion:

When a decimal number is POKEd into memory, only the integer part is saved. The decimal part of the number is dropped off and lost. Since some of the above data is to the tenths place, all of the data must be multiplied by ten to make them all integers. If some were to the hundredths place, all would have to be multiplied by 100. After the date is multiplied by 10, a second problem arises: the data is greater than 255. Since the data is over 255, each will take two bytes of memory. Some of the data is still less than 256 and would fit into one byte, but to make the data easier to read back into variables, two bytes will be used for each.

Data items larger than 255 must be converted so that they can be POKEd into two bytes. The two bytes that will make up a data item will be called the "high byte" and the "low byte." The high byte is found by taking the integer part of the data item divided by 256. The low byte is the remainder of the data item divided by 256. If a data item is less than 256, the high byte will be zero and the low byte the item itself.

program 2:
(C4P2)

```
10   HOME
20   HIMEM:16400
30   D$ = CHR$(4)
40   FOR I = 1 TO 25
50   READ A
60   A = A * 10
70   VTAB 10 : HTAB 15 : PRINT I
80   L = A
90   H = 0
100  IF A < 256 THEN 130
110  H = INT(A / 256)
120  L = (A / 256 - INT(A / 256)) * 256
130  POKE(16401 + (I - 1) * 2),H
140  POKE(16402 + (I - 1 ) * 2),L
150  NEXT I
160  DATA 17.6,12.4,32,9.2,101.5,66.1,98.6,200,7.2
170  DATA 12.8,21.1,43.7,182,0,76.5,57.1,81.1,18.8
180  DATA 13.3,12.2,6.3,8.2,11.3,182.6,133.6
190  PRINT D$;"BSAVE BFR,A16401,L50"
200  END
```

program explanation:

Line 20 -- set HIMEM to 16400.
Line 50 — read a data item.
Line 60 - multiply data by 10 to make them all integers.
Line 100 — check if data is less than 256.
Line 110 - if data is more than 255, find the integer part of the data item divided by 256.
Line 120 -- find the remainder of the data item divided by 256.
Line 130 — POKEs the high byte.
Line 140 - POKEs the low byte.
Line 190 - saves the binary data file. The length is 50 since each data item takes two bytes.

problem 3: Write a program that will save the following five alpha data items in a binary file. The data items are "ABC", "XYZ", "QWRTY", "MPLS", "RNBIA".

problem discussion: The first step in making alpha data into a binary file is to find the data item with the most characters, in this case five. Each item will then take up five bytes of storage to make reading the item back into variables easier. The characters in the string are converted to their numeric equivalent with the ASC function. If an item is less than five characters, the remaining bytes it will take up will be POKEd with 32, the equivalent of a space.

program 3:
(C4P3)

```
10   HOME
20   HIMEM:16400
30   D$ = CHR$(4)
40   FOR I = 1 TO 5
50   READ A$
60   FOR J = 1 TO LEN(A$)
70   B$ = MID$(A$,J,1)
```

```
80   B = ASC(B$)
90   P = 16401 + (J - 1) + (I - 1) * 5
100  POKE P,B
110  NEXT J
120  IF LEN(A$) = 5 THEN 170
130  FOR J = LEN(A$) + 1 TO 5
140  P = 16401 + (J - 1) + (I - 1) * 5
150  POKE P,32
160  NEXT J
170  NEXT I
180  DATA "ABC","XYZ","QWRTY","MPLS","RNBIA"
190  PRINT D$;"BSAVE SBF,A16401,L25"
200  END
```

program explanation:

Line 70 -- B$ is one character of the string A$.
Line 80 — B is the decimal equivalent of B$.
Line 90 — P is the memory address where B will be POKEd.
Line 100 — B is POKEd into Apple memory address P.
Line 120 - 160 -- if the length of A$ is less than five characters, 32 (space) is POKEd in the remaining bytes.
Line 190 — the binary file SBF with length 25 is saved.

EXERCISE SET 4A

1. Write the statement that will POKE 68 into memory location 17000.

2. What happens when a number greater than 255 is POKEd into the computer memory?

3. Define "binary file."

4. What would be the high byte and low byte of the number 23865?

5. What is the largest number that can be POKEd into two bytes?

6. Write the two statements that will POKE 862 into memory locations 17500 and 17501.

7. Why are spaces placed in any unused bytes in a binary string data item?

4.3 READING A BINARY DATA FILE

To read a binary data file, the file command BLOAD and the program statement PEEK are needed.

PEEK Statement

general form:

```
ln  x = PEEK (y)
```

where:

ln is any legal line number.
x is any numeric variable.
y is an Apple memory location.

example:

```
60  L = PEEK (17000)
70  P = PEEK (768)
```

purpose:

To read the content of a memory location and store the contents in a variable.

BLOAD Statement

general form:

```
ln  PRINT D$;"BLOAD name, Ax"
```

where:

ln is any legal line number.
D$ is a control-D.
name is a binary file name.
x is the address to start loading the file.

example:

```
80  PRINT D$;"BLOAD AJT,A31000"
90  PRINT D$;"BLOAD NAUC,A$4000"
```

purpose:

To place the contents of a binary file into the computer's memory, starting at a specific location.

problem 4: Write a program that will print the fifth, tenth, twentieth, and twenty-fifth data items in the binary file "BFR" created by program 2.

program 4:
(C4P4)

```
10  HOME
20  HIMEM:16400
30  D$ = CHR$(4)
40  PRINT D$;"BLOAD BFR,A16401"
50  FOR I = 5 TO 25 STEP 5
60  H = PEEK (16401 + (I - 1) * 2)
70  L = PEEK (16402 + (I - 1) * 2)
80  N = (H * 256 + L)/10
90  PRINT I,N
100  NEXT I
110  END
```

program:

Line 40 -- loads the file BFR off of the diskette and into the computer's memory.
Line 60 — the high byte is found.
Line 70 — the low byte is found.
Line 80 — the number is calculated. The number is divided by 10, because all of them were multiplied by 10 in program 2.

problem 5: Write a program that will print the second and third data items in the binary file "SBF" created by program 3.

program 5:
(C4P5)

```
10  HOME
20  HIMEM:16400
30  D$ = CHR$(4)
40  PRINT D$;"BLOAD SBF,A16401"
50  A = 2
60  S$ = ""
70  FOR I = 1 TO 5
80  P = 16401 + (A - 1) * 5 + (I - 1)
90  Q = PEEK(P)
100  S$ = S$ + CHR$(Q)
110  NEXT I
120  PRINT S$
130  A = A + 1
140  IF A = 4 THEN 160
150  GOTO 60
160  END
```

program explanation:

Line 40 -- the binary file SBF is loaded into the computer's memory, starting at address 16401.
Line 80 -- the memory location to PEEK is found.
Line 90 — the memory location is PEEKed.
Line 100 -- the number in the memory location PEEKed is converted to the alpha equivalent and added to the string S$.
Line 120 — the data item is printed.

EXERCISE SET 4B

1. Write a statement that will PEEK memory location 17000.

2. If the high byte of a number is in memory location 16384 and the low byte is in memory location 16385, write the statement needed to find the number.

3. Can a binary data file be run?

4. Write a program that will read any of the data items of a binary data file. The file starts at memory location 17501 and each numeric data item requires two bytes of memory. (Use the Binary Data File SBF from program 3 to test your program.)

5. Write a program that will read any of the data items in a binary file that starts at memory location 31501 and contains alpha data that requires ten bytes of memory each.

Review - Chapter 4

Chapter 4 explained how to create and read a binary data file. The following statements were introduced:

BLOAD Statement

general form: PRINT D$;"BLOAD B,A20000"

The BLOAD statement is used to place a binary file into the computer's memory from a diskette at the location specified by the A parameter.

BSAVE Statement

general form: PRINT D$;"BSAVE B,A16000,L60"

The BSAVE statement is used to save a group of memory locations starting at the address specified by the A parameter and to save the number of memory locations specified by the L parameter.

PEEK Statement

general form: L = PEEK (176)

The PEEK statement is used to read the contents of a memory location.

POKE Statement

general form: POKE 176,8

The POKE statement is used to change the contents of a memory location.

The problems and programs presented in the chapter should be studied, entered into the computer, and run. To make other binary data files, the same programs could be used. The only changes would be the data, the starting address, and the length of the files.

SAMPLE PROGRAMS

The following programs show how binary data files could be used in an application. The program will determine the weight range for 15-18 year-olds, given their sex, age, and height.

PROGRAM C4P6

The first program is used to create the binary data file to be used in the main program. The first data that is POKEd into memory is the height and weight for 15-year-old females and the last data POKEd is the height and weight for 18-year-old males.

```
10  HOME
20  HIMEM: 20000
30  D$ = CHR$ (4)
40  FOR I = 1 TO 8
50  IF I< 5 THEN VTAB 10: PRINT "DATA FOR FEMALE AGE ";I + 14;" BEING
    ENTERED"
60  IF I >4 THEN VTAB 10: CALL - 868: PRINT "DATA FOR MALE AGE ";I +
    10;" BEING ENTERED"
70  FOR J = 1 TO 14
80  H = 0
90  READ A
100 L = A * 10
110 IF L< 256 THEN 140
120 H = INT (L/256)
130 L = (L/256 - INT (L/256)) * 256
140 POKE (20000 + (I - 1) * 28 + (J - 1) * 2),H
150 POKE (20000 + (I - 1) * 28 + (J - 1) * 2 + 1),L
160 NEXT J
170 NEXT I
180 PRINT D$;"BSAVE SAMPLEC4P6,A20000,L120"
190 DATA 59.1,89,61.1,97.4,62.1,105.1,63.4,113.5,64.9,123.9,66.2,138.1,67.6,155.2
200 DATA 59.4,91.8,61.5,100.9,62.4,108.4,63.9,117,65.2,127.2,66.5,141.1,67.7,157.7
210 DATA 59.4,93.9,61.5,102.8,62.6,110.4,64,119.1,65.4,129.6,66.7,143.3,67.8,159.5
220 DATA 59.4,94.5,61.5,103.5,62.6,111.2,64,119.9,65.4,130.8,66.7,114.5,67.8,160.7
230 DATA 59.7,91.3,62.1,99.4,63.9,108.2,66.1,120.1,68.1,135,69.6,147.8,71.6,161.6
240 DATA 61.6,103.4,64.1,111,65.8,118.7,67.8,129.7,69.5,114.4,70.7,157.3,73.1,170.5
250 DATA 62.6,110.5,65.2,117.5,66.8,124.5,68.4,136.2,70.1,151.4,71.5,164.6,73.5,175.6
260 DATA 62.8,113,65.5,120,67,127.1,68.7,139,70.4,155.7,71.8,169,73.9,179
270 END
```

PROGRAM C4P7

The following program is the main program that uses the binary data file created by the program C4P6.

```
10  HOME
20  HIMEM: 20000
30  D$ - CHR$ (4)
32  HOME : VTAB 4: PRINT "THIS PROGRAM WILL DETERMINE THE": PRINT :
    PRINT "WEIGHT RANGE FOR PEOPLE AGED 15-18."
35  VTAB 20: HTAB 5: PRINT "ENTER A SPACE TO CONTINUE";:POKE - 16368,0:
    GET A$: PRINT
40  VTAB 3: HTAB 5: PRINT "ENTER YOUR SEX (M OR F)";: CALL - 958
50  INPUT S$:S$ = LEFT$ (S$,1): IF S$<>"M" AND S$<>"F" THEN 40
60  VTAB 7: HTAB 5: PRINT "ENTER YOUR AGE (15-18)";: CALL - 958
70  INPUT A$:A = VAL (A$): IF A< 15 OR   A > 18 THEN 60
80  VTAB 11: HTAB 5: PRINT "ENTER YOUR HEIGHT (INCHES)";: CALL - 958
90  INPUT H$:H = VAL (H$): IF H<= 0 THEN 80
100 PRINT D$;"BLOAD SAMPLEC4P6,A20000"
110 A1 = A - 15: IF S$ = "M" THEN A1 = A1 + 4
115 S1$ = "FEMALE": IF S$ = "M" THEN S1$ = "MALE"
120 W1 = 0
130 AD = 20000 + A1 * 28
140 FOR I = 1 TO 7
150 HE = PEEK (AD + (I - 1) * 4) * 256 + PEEK (AD + 1 + (I - 1) * 4)
155 HE = HE / 10
160 W2 = PEEK (AD + 2 + (I - 1) * 4) * 256 + PEEK (AD + 3 + (I - 1) * 4)
165 W2 = W2 / 10
170 IF H< HE AND W1 = 0 THEN 210
180 IF H< HE THEN 220
190 W1 = W2
200 NEXT I
210 HOME : VTAB 5: PRINT "SORRY, I DO NOT HAVE DATA FOR ": PRINT : PRINT
    "A ";S1$;" AGED ";A;" WITH": PRINT : PRINT "A HEIGHT OF ";H;" INCHES": GOTO
    230
220 HOME : VTAB 5: PRINT "I BELIEVE A ";S1$;"  AGED ";A;" WITH": PRINT : PRINT
    "A HEIGHT OF ";H;" INCHES": PRINT : PRINT "SHOULD WEIGH ";W1;" TO ";W2;"
    POUNDS."
230 END
```

Review Quiz - Chapter 4

1. Write a statement to POKE 163 into memory location 24092.

2. Which of the following BSAVE statements are correctly written?

   a. PRINT CHR$ (4);"BSAVE ABC,L123,A16000"
   b. PRINT "BSAVE CDE,A16000,L16"
   c. PRINT CHR$(4);"BSAVE XYZ,A17000,L20"
   d. PRINT D$;"BSAVE";F$;"A17000,L20"
   e. PRINT D$;"BSAVE";F$

3. Write a statement to PEEK at memory location 23891.

4. Which of the following BLOAD statements are correctly written?

   a. PRINT D$;"BLOAD ABC"
   b. PRINT CHR$ (4);"BLOAD XYZ,L60"
   c. PRINT CHR$ (4);"BLOAD QWE,A17362"
   d. PRINT CHR$ (4);"BLOAD";F$;",A16742"
   e. PRINT D$;"BLOAD CE,A17171,L71"

CHAPTER 5

SPECIAL FEATURES AND UTILITY PROGRAMS

5.1 EXEC FILES

Unlike other files that contain programs or data, EXEC files contain commands or program statements. The commands or statements are written on the file just as you would normally type them on the keyboard. A program that creates this type of file simply opens a file and writes each command or statement to the file. In order to execute the commands and statements in the file at a later time, type: EXEC name. The following file commands are used to create EXEC files:

```
OPEN
WRITE
CLOSE
EXEC
```

general form:

```
EXEC name (immediate command)
ln  PRINT D$;"EXEC name" (as used in a program)
```

where:

ln is a legal line number.
name is any file name.

example:

```
EXEC FILE6
46  PRINT D$;"EXEC FILE6"
```

problem 1: Create a program that will allow the user to type in Apple commands for later execution.

program 1:
(C5P1)

```
10  D$ = CHR$ (4)
20  PRINT D$;"OPEN STATEMENTS"
30  PRINT "HOW MANY COMMANDS WILL BE ENTERED"
40  INPUT N
50  PRINT "ENTER A COMMAND AFTER EACH QUESTION
    MARK"
60  FOR X = 1 TO N
70  INPUT A$
80  PRINT D$;"WRITE STATEMENTS"
90  PRINT A$
100  PRINT D$
110  NEXT X
120  PRINT D$;"CLOSE STATEMENTS"
130  PRINT D$
140  END
```

program description: The commands written to the file STATEMENTS can now be executed in either of the following two ways:

In immediate mode by typing

```
EXEC STATEMENTS
```

or by putting the following statement in a program

```
123 PRINT D$;"EXEC STATEMENTS"
```

problem 2: Create a program that creates an EXEC file that will load, list, and run a program named 'HT'. The program HT is saved on the diskette.

program 2:
(C5P2)

```
10  D$ = CHR$ (4)
20  PRINT D$;"OPEN STATEMENTS1"
30  PRINT D$;"WRITE STATEMENTS1"
40  READ A$
50  PRINT A$
60  IF A$ = "RUN" THEN 80
70  GOTO 40
80  PRINT D$;"CLOSE STATEMENTS1"
90  DATA "LOAD HT", "LIST", "RUN"
100  END
```

program description: The EXEC file can now be executed by typing either

```
EXEC STATEMENTS1
```

or putting the following statement in a program:

```
246 PRINT D $;"EXEC STATEMENTS1"
```

5.2 CHECKING FOR ILLEGAL FILES

Many times when writing programs that use files, you want to ask the user to enter the name of the file to be used in the program. If the user enters a file name that is improper, the program will terminate with an error. The error is usually "END OF DATA" because the file did not exist and the program errored when it tried to read from the file. What happened is that the OPEN statement did not find a file with the name entered, so it added the file name to the CATALOG. The READ statement prepared the file for reading, but when the INPUT statement was executed no data was found and the program terminated. When this happens the CATALOG of the diskette will list all of the improper file names that had been entered. The file names will each take up one sector of storage and files will have to be deleted to clean up the CATALOG listing. By using the APPLESOFT statement ONERR this problem can be eliminated.

ONERR statement

Intercepts errors that terminate programs. There is more about ONERR in Section 5.7.

general form: ln ONERR GOTO ln

where: ln is any legal line number

example: 10 ONERR GOTO 1000

Try the following program:

problem 3: Create a program that will determine whether a file is a legal file without erroring.

program 3:
(C5P3)

```
10 D$ = CHR$ (4)
20 PRINT "ENTER YOUR FILE NAME"
30 INPUT F$
40 ONERR  GOTO 100
50 PRINT D$;"OPEN ";F$
60 PRINT D$;"READ ";F$
70 INPUT A$
80 PRINT A$
90 GOTO 140
100 PRINT D$;"CLOSE ";F$
```

```
110 PRINT D$;"DELETE ";F$
120 PRINT "IMPROPER FILE NAME"
130 GOTO 20
140 PRINT D$;"CLOSE ";F$
150 END
```

program description: Line 40 - if an error occurs, control is transferred to line 100.
Line 100 - the file is closed.
Line 110 - the improper file is deleted from the CATALOG.

This program takes many steps to determine whether a file name is proper. A quicker way to do this program is with the VERIFY statement.

VERIFY statement:

general form: ln PRINT D$;"VERIFY name"

where: ln is any legal line number
D$ is a control D
name is the file name.

example: 45 PRINT D$;"VERIFY FILE99"

The VERIFY statement checks the CATALOG for the file indicated. If the file is not found, the statement errors. If the file name is found, the VERIFY statement checks to see if any of the file's sectors are bad.

Try the following program:

problem 4: Rewrite program 3, but use the VERIFY statement.

program 4:
(C5P4)

```
10 D$ = CHR$ (4)
20 PRINT "ENTER YOUR FILE NAME"
30 INPUT F$
40 ONERR  GOTO 110
50 PRINT D$;"VERIFY ";F$
60 PRINT D$;"OPEN ";F$
70 PRINT D$;"READ ";F$
80 INPUT A$
90 PRINT A$
100 GOTO 130
110 PRINT "IMPROPER FILE"
120 GOTO 20
130 PRINT D$;"CLOSE ";F$
140 END
```

program description: Line 40 - if an error occurs, control is transferred to line 110.
Line 50 - the file is verified.

DISCUSSION OF PROGRAM 3 AND PROGRAM 4

Program 4 is better than program 3 not only because it uses fewer statements, but also because it does not use a DELETE statement. When program 3 is executed, errors other than "END OF DATA" could occur from reading the file. If some other error occurs, the program control is still transferred to the CLOSE and DELETE statements. If this happens, you could be deleting a very important file or program.

With very long files the VERIFY statement could take up to ten seconds to verify the file. The program will run faster and almost as well if another file statement, such as UNLOCK, is used in place of VERIFY.

5.3 USING MORE THAN ONE FILE

Many times more than one file is needed at a time in a program. More than one active file can be used simultaneously in a program just by using more than one OPEN statement. When the computer is first turned on it allows the use of a maximum of three active files. If more files than this are needed, the MAXFILES statement will have to be used.

MAXFILES statement:

general form: ln MAXFILES x

where: ln is a legal line number.
x is the number of files needed. The range is from 1 to 16.

example:

```
36 MAXFILES 5
43 MAXFILES 2
```

If the MAXFILES statement is used, it must be one of the first statements in the program. If MAXFILES is used later in the program, the program and some of the string variables will be destroyed.

5.4 KEEPING TRACK OF THE NUMBER OF FIELDS WRITTEN

In Chapter 3 it was explained that record zero is a good place to keep track of the number of records that have been written to a random access file, but what about sequential files? It would be great if the first thing read from a sequential file were the number of fields in the file. This can be done if you remember the problems that can occur when rewriting fields in sequential files.

example: Suppose eight items were written to a sequential file; it would be easy to write the number 8 in the first field and the eight items after that. Now suppose three more items were to be added to the file. By using the APPEND statement three more fields could be added to the file, then the file closed, reopened, and the number 11 written to the first field in the file. The problem with this is that the 11 takes two bytes of storage and the 8 only one byte; thus we have destroyed part of the second field. This problem can be eliminated by making sure the same number of characters is always written to the first field in the file.

The steps in doing this are listed below:

Step 1: Change the number, 8 for example, to a string:

```
10 A= 8

20 A$=STR$(8)
```

Step 2: Add four zeros to the beginning of A$:

```
30 A$="0000" + A$
```

A$ is now equal to "00008".

Step 3: Set the right four characters of A$ equal to A$:

```
40 A$=RIGHT$(A$,4)
```

A$ is now equal to "0008".

Step 4: Write A$ to the first field of the file.

To find out how many items have been written to the file, read the first field of the file as A$ and change it to a value with the following statement:

```
A=VAL(A$)
```

Suppose more items, three for example, were to be added to the file; the items could be appended as in Chapter 2. The first field of the file would now have to be rewritten to indicate the new total number of fields used in the file.

The steps in doing this are listed below:

Step 1: Be sure the file is closed after the appending of the items is finished.

Step 2: Reopen the file, read the first field, and close the file:

```
100 PRINT D$;"OPEN ";F$
110 PRINT D$;"READ ";F$
120 INPUT A$
130 PRINT D$;"CLOSE ";F$
```

A$ will equal "0008".

Step 3: Change A$ to a value and add three to it:

```
140 A=VAL(A$) + 3
```

A is equal to 11.

Step 4: Change A to the string A$, add four zeros to A$, and set A$ to the right four characters in A$:

```
160 A$=RIGHT$("0000"+STR$(A),4)
```

Step 5: Reopen the file again, print A$ to the first field in the file, and close the file:

```
170 PRINT D$;"OPEN ":F$
180 PRINT A$
190 PRINT D$;"CLOSE"
```

The first field in the file now contains the number of fields used in the file.

EXERCISE SET 5A

1. Explain the command MAXFILES 5.

2. What command should be used to check if a file has been written to the diskette correctly?

3. What is the maximum number of files that can be used in a program at one time?

4. Write a program that will write out to a sequential file the following three commands: RUN PROG1, LIST, DELETE PROG1. Include an EXEC DOS statement that will execute the above commands. Write a program called PROG1 that will generate a simple solution when the EXEC statement is executed.

5. What is the default value for the number of active files in a program?

5.5 UTILITY PROGRAMS

Several utility programs have been written that will make file manipulation easier. The following programs are on the Programmers's Aid Volume 1 diskette and support booklet, published by MECC. The support booklet will aid you in the use of the program and can be ordered from MECC.

Binary File Info

This program will give the beginning memory address and the length of a binary file. The beginning address must be known to BLOAD a binary file. Both the beginning address and the length must be known to BSAVE the binary file.

Binary File to FP

This program will convert a binary file to an APPLESOFT program so that it can be loaded and listed.

FP to Text

This program will convert the APPLESOFT program to a TEXT file that can be read by a different program. This text file can now be edited by the word processor Applewriter II and then turned back into a program with the EXEC command.

Free Space

This program will print out the percentage of sectors that have not been filled on a given diskette.

Random Editor

This program can be used to make changes to a random file.

Sequential Editor

This program can be used to make changes to a sequential file.

Text File

This program will list out any given TEXT file.

Another program with several utility options is the file named 'FID' on the DOS 3.3 SYSTEM MASTER DISKETTE that comes packaged with new Apple Computers. To execute this file, type: BRUN FID

The available options are:

1. copy files
2. catalog
3. space on disk
4. unlock files
5. lock files
6. delete files
7. reset slot and drive
8. verify files

5.6 MAP OF COMMANDS

Several DOS commands have been discussed in this booklet. The following chart indicates the types of files each command will work for:

DOS Command	Sequential Text File	Random Access Text File	Binary File
APPEND	X		
BLOAD			X
BSAVE			X
CLOSE	X	X	
DELETE	X	X	X
EXEC	X		
LOCK	X	X	X
OPEN	X	X	
POSITION	X		
READ	X	X	
RENAME	X	X	X
UNLOCK	X	X	X
VERIFY	X	X	X
WRITE	X	X	

5.7 ERROR PROCESSING

By using the APPLESOFT's statement ONERR GOTO, you can create error-handling routines that deal with DOS messages that would normally interrupt your program. When a DOS error occurs following an ONERR GOTO statement, a code number for the type of error is stored in decimal memory location 222. The statement to print the code is PRINT PEEK(222).

example:

```
10   ONERR GOTO 40
40   PRINT PEEK(222)
```

The code number for errors associated with text and binary files are as follows:

Code	Error Message	Common Cause
2,3	Range Error	Command parameter too large (slot number, drive number, maximum byte number, record length, number of bytes, record number)
4	Write Protected	Write protect tab on diskette
5	End of Data	Reading beyond the end of a text file
6	File Not Found	File misspelled or not on diskette
8	I/O Error	Door open or diskette not initialized
9	Diskette Full	Too many files on diskette
10	File Locked	Attempt to over-write a locked file
12	No Buffers Available	Too many text files open
13	File Type Mismatch	Diskette file doesn't match command (i.e., trying to LOAD a binary file)
14	Program Too Large	Insufficient Apple memory available
15	Not Direct Command	Command is not an immediate command and must be used in a program

The Minnesota Educational Computing Consortium is an organization established in 1973 to assist Minnesota schools and colleges in implementing educational computing. MECC provides a variety of services to education, including 1) development and distribution of computer software; 2) in-service training for educators and development of materials for conducting training; 3) educational computing assistance through newsletters and computer purchase contracts; and 4) management information services, including the development and maintenance of statewide payroll/personnel and financial accounting software and administrative computer packages. MECC's knowledge and expertise in the educational computing field comes from a decade of working with and providing leadership for hundreds of local educators on a daily basis.

- **MECC Educational Computing Catalog**
  Catalogs containing instructional computing courseware, all-purpose training materials, and administrative software are published twice each year and are distributed at no charge. To request a catalog, write or call MECC Distribution (Telephone: 612/481-3527).

- **MECC Memberships**
  Non-Minnesota non-profit educational institutions may obtain annual service agreements with MECC which qualify them to obtain MECC courseware and training at special reduced prices. For up-to-date pricing and procedural information on these memberships, write or call MECC Institutional Memberships (Telephone: 612/481-3512).

- **Training Programs**
  MECC staff conducts educational computing workshops for educators throughout the United States. For information on workshop schedules or to arrange a special training activity, write or call MECC User Services (Telephone: 612/481-3651).

- **Administrative Software**
  MECC provides a variety of quality administrative microcomputer packages. For information on available packages and on training and maintenance contracts, write or call MECC-MIS (Telephone: 612/481-3531).

- **MECC Network Newsletter**
  Published regularly throughout the school year, MECC's newsletter focuses on MECC activities, training materials, and educational courseware. To obtain, write or call indicating your interest in the MECC Network Newsletter (Telephone: 612/481-3612).

- **Help Line**
  If you have any problems using MECC software:
  1) make note of the name and version number of the product;
  2) note the brand and model as well as the type of printer card used if the problem concerns the printer;
  3) write or call the Help Line to describe the problem (612/481-3660).

MECC
3490 Lexington Avenue North
St. Paul, MN 55112
(General Information: 612/481-3500)